Annexe A : Code source

Orphée Antoniadis

Table des matières

Table des matières **1**

I Structure du projet **2**

II /common **3**
II.I /common/src/fs.rs . . . 3
II.II /common/src/keyboard.rs . . . 4
II.III /common/src/lib.rs . . . 5
II.IV /common/src/string.rs . . . 5
II.V /common/src/syscall.rs . . . 6
II.VI /common/src/vga.rs . . . 7
II.VII /common/Cargo.toml . . . 9

III /doc/splash.txt **10**

IV /grub/menu.lst **11**

V /kernel **12**
V.I /kernel/src/bootstrap_asm.s . . . 12
V.II /kernel/src/const.inc . . . 15
V.III /kernel/src/fs.rs . . . 15
V.IV /kernel/src/gdt_asm.s . . . 21
V.V /kernel/src/gdt.rs . . . 22
V.VI /kernel/src/ide.rs . . . 25
V.VII /kernel/src/idt_asm.s . . . 27
V.VIII /kernel/src/idt.rs . . . 29
V.IX /kernel/src/kernel.ld . . . 35
V.X /kernel/src/kernel.rs . . . 37
V.XI /kernel/src/keyboard.rs . . . 39
V.XII /kernel/src/kheap.rs . . . 41
V.XIII /kernel/src/Makefile . . . 48
V.XIV /kernel/src/multiboot.rs . . . 50
V.XV /kernel/src/paging_asm.s . . . 52
V.XVI /kernel/src/paging.rs . . . 54
V.XVII /kernel/src/pic.rs . . . 60
V.XVIII /kernel/src/pio_asm.s . . . 61
V.XIX /kernel/src/pio.rs . . . 63
V.XX /kernel/src/syscall_asm.s . . . 64
V.XXI /kernel/src/syscall.rs . . . 65
V.XXII /kernel/src/task_asm.s . . . 69
V.XXIII /kernel/src/task.rs . . . 69
V.XXIV /kernel/src/timer.rs . . . 74
V.XXV /kernel/src/vga.rs . . . 75
V.XXVI /kernel/src/x86.rs . . . 79
V.XXVII /kernel/Cargo.toml . . . 80
V.XXVIII /kernel/i386-rust_os.json . . . 81
V.XXIX /kernel/Makefile . . . 81

VI /tools **83**
VI.I /tools/MircroFS/src/micro_fs/add.rs . . . 83
VI.II /tools/MircroFS/src/micro_fs/create.rs . . . 83
VI.III /tools/MircroFS/src/micro_fs/del.rs . . . 83
VI.IV /tools/MircroFS/src/micro_fs/info.rs . . . 83
VI.V /tools/MircroFS/src/micro_fs/list.rs . . . 83
VI.VI /tools/MircroFS/src/micro_fs/mod.rs . . . 83
VI.VII /tools/MircroFS/src/micro_fs/save.rs . . . 83
VI.VIII /tools/MircroFS/src/micro_fs/utils.rs . . . 83
VI.IX /tools/MircroFS/src/cli.yml . . . 83
VI.X /tools/MircroFS/src/lib.rs . . . 83
VI.XI /tools/MircroFS/src/main.rs . . . 84
VI.XII /tools/MircroFS/tests/block_size_1.rs . . . 84
VI.XIII /tools/MircroFS/Cargo.toml . . . 84

VII /user **85**
VII.I /user/demo/src/demo.rs . . . 85
VII.II /user/demo/src/hello.rs . . . 86
VII.III /user/demo/src/shell.rs . . . 86
VII.IV /user/skeleton/src/skeleton.rs . . . 87
VII.V /user/skeleton/Cargo.toml . . . 87
VII.VI /user/skeleton/Makefile . . . 87
VII.VII /user/splash/src/splash.rs . . . 88
VII.VIII /user/ulibc/src/curses.rs . . . 88
VII.IX /user/ulibc/src/io.rs . . . 90
VII.X /user/ulibc/src/mem.rs . . . 93
VII.XI /user/ulibc/src/ulibc.rs . . . 98
VII.XII /user/ulibc/Cargo.toml . . . 98
VII.XIII /user/app.ld . . . 99
VII.XIV /user/entrypoint_asm.s . . . 99
VII.XV /user/i386-app.json . . . 100
VII.XVI /user/Makefile . . . 100
VII.XVII /user/syscall_asm.s . . . 101

VIII /Makefile **103**

I Structure du projet

```
common
├── src
│   ├── fs.rs
│   ├── keyboard.rs
│   ├── lib.rs
│   ├── string.rs
│   ├── syscall.rs
│   └── vga.rs
└── Cargo.toml
doc
└── splash.txt
grub
├── menu.lst
└── stage2_eltorito
kernel
├── src
│   ├── bootstrap_asm.s
│   ├── const.inc
│   ├── fs.rs
│   ├── gdt_asm.s
│   ├── gdt.rs
│   ├── ide.rs
│   ├── idt_asm.s
│   ├── idt.rs
│   ├── kernel.ld
│   ├── kernel.rs
│   ├── keyboard.rs
│   ├── kheap.rs
│   ├── Makefile
│   ├── multiboot.rs
│   ├── paging_asm.s
│   ├── paging.rs
│   ├── pic.rs
│   ├── pio_asm.s
│   ├── pio.rs
│   ├── syscall_asm.s
│   ├── syscall.rs
│   ├── task_asm.s
│   ├── task.rs
│   ├── timer.rs
│   ├── vga.rs
│   └── x86.rs
├── Cargo.toml
├── i386-rust_os.json
└── Makefile
tools
└── MicroFS
    ├── src
    │   ├── micro_fs
    │   │   ├── add.rs
    │   │   ├── create.rs
    │   │   ├── del.rs
    │   │   ├── info.rs
    │   │   ├── list.rs
    │   │   ├── mod.rs
    │   │   ├── save.rs
    │   │   └── utils.rs
    │   ├── cli.yml
    │   ├── lib.rs
    │   └── main.rs
    ├── tests
    │   ├── block_size_1.rs
    │   ├── test1.text
    │   └── test2.text
    └── Cargo.toml
user
├── demo
├── hello
├── shell
├── skeleton
│   ├── src
│   │   └── skeleton.rs
│   ├── Cargo.toml
│   └── Makefile
├── splash
├── ulibc
│   ├── src
│   │   ├── curses.rs
│   │   ├── io.rs
│   │   ├── mem.rs
│   │   └── ulibc.rs
│   └── Cargo.toml
├── app.ld
├── entrypoint_asm.s
├── i386-app.json
├── Makefile
└── syscall_asm.s
Makefile
```

II /common

II.I /common/src/fs.rs

```rust
pub const MAX_FILENAME_LENGTH: usize = 26;

#[derive(Debug, Clone, Copy)]
#[repr(C)]
pub struct Stat {
    pub name: [u8;MAX_FILENAME_LENGTH],
    pub size: usize,
    pub entry_offset: u16,
    pub start: usize
}

#[derive(Debug, Clone, Copy)]
#[repr(C)]
pub struct FileIterator {
    pub sector: u32,
    pub offset: usize
}

impl Stat {
    pub fn null() -> Stat {
        Stat {
            name: [0;MAX_FILENAME_LENGTH],
            size: 0,
            entry_offset: 0,
            start: 0
        }
    }

    pub fn as_ptr(&mut self) -> *const Stat {
        self as *const Stat
    }
}

impl FileIterator {
    pub fn null() -> FileIterator {
        FileIterator {
            sector: 0,
            offset: 0
        }
    }

    pub fn as_ptr(&mut self) -> *const FileIterator {
        self as *const FileIterator
    }
```

```
}
```

II.II /common/src/keyboard.rs

```
// Ascii charset hex
pub const NUL: char = '\0';
pub const BACKSPACE: char = 0x8 as char;
pub const NAK: char = 0x15 as char;
pub const E_ACUTE: char = 0x82 as char;
pub const A_GRAVE: char = 0x85 as char;
pub const C_CEDILLA: char = 0x87 as char;
pub const E_GRAVE: char = 0x8a as char;
pub const U_GRAVE: char = 0x97 as char;
pub const POUND: char = 0x9c as char;

// Keys codes
pub const CTRL: u8 = 29;
pub const LEFT_SHIFT: u8 = 42;
pub const RIGHT_SHIFT: u8 = 54;
pub const ALT: u8 = 56;
pub const CAPS_LOCK: u8 = 58;
pub const UP_KEY: u8 = 72;
pub const LEFT_KEY: u8 = 75;
pub const RIGHT_KEY: u8 = 77;
pub const DOWN_KEY: u8 = 80;
pub const LEFT_CMD: u8 = 91;
pub const RIGHT_CMD: u8 = 92;

pub const KEY_MAP: [char;93] = [
    NUL, NUL, '&', E_ACUTE, '"', '\'', '(', NAK, E_GRAVE, '!', C_CEDILLA, A_GRAVE, ')', '-', BACKSPACE,
    '\t', 'a', 'z', 'e', 'r', 't', 'y', 'u', 'i', 'o', 'p', '^', '$', '\n',
    NUL, 'q', 's', 'd', 'f', 'g', 'h', 'j', 'k', 'l', 'm', U_GRAVE, '@',
    NUL, '`', 'w', 'x', 'c', 'v', 'b', 'n', ',', ';', ':', '=', NUL,
    NUL, NUL, ' ', NUL, ' ', NUL, NUL, NUL, NUL, NUL, NUL, NUL, NUL,
    NUL, NUL, NUL, NUL, NUL, NUL, NUL, NUL, NUL, NUL, NUL, NUL, NUL,
    NUL, NUL, NUL, NUL, NUL, '<', NUL, NUL, NUL, NUL, NUL, NUL
];

pub const SHIFT_KEY_MAP: [char;93] = [
    NUL, NUL, '1', '2', '3', '4', '5', '6', '7', '8', '9', '0', '°', '_', BACKSPACE,
    '\t', 'A', 'Z', 'E', 'R', 'T', 'Y', 'U', 'I', 'O', 'P', NUL, '*', '\n',
    NUL, 'Q', 'S', 'D', 'F', 'G', 'H', 'J', 'K', 'L', 'M', '%', '#',
    NUL, POUND, 'W', 'X', 'C', 'V', 'B', 'N', '?', '.', '/', '+', NUL,
    NUL, NUL, ' ', NUL, ' ', NUL, NUL, NUL, NUL, NUL, NUL, NUL, NUL,
```

```
    NUL, NUL, NUL, NUL, NUL, NUL, NUL, NUL, NUL, NUL, NUL, NUL, NUL,
    NUL, NUL, NUL, NUL, NUL, '>', NUL, NUL, NUL, NUL, NUL, NUL
];
```

II.III /common/src/lib.rs

```
#![feature(const_fn)]
#![no_std]

mod syscall;
mod string;
mod fs;
mod keyboard;
mod vga;

pub use syscall::*;
pub use string::*;
pub use fs::*;
pub use keyboard::*;
pub use vga::*;
```

II.IV /common/src/string.rs

```
use core::slice::from_raw_parts;
use core::str::from_utf8;

pub const MAX_STR_LEN : usize = 256;

#[derive(Debug, Clone, Copy)]
#[repr(C)]
pub struct String {
    pub bytes_ptr: u32,
    pub len: usize
}

pub fn bytes_to_str(bytes: &[u8]) -> &str {
    let mut cnt = 0;
    for &byte in bytes {
        if byte == 0 {
            break;
        } else if byte > 0x7f {
            return "\0";
        }
        cnt += 1;
    }
    from_utf8(&bytes[0..cnt]).expect("Found invalid UTF-8")
}
```

```rust

impl String {
    pub fn new(s: &str) -> String {
        unsafe {
            String {
                bytes_ptr: &from_raw_parts(s.as_ptr(), s.len())[0] as *const u8 as u32,
                len: s.len()
            }
        }
    }

    pub fn to_string(&mut self) -> &str {
        unsafe {
            let addr = self.bytes_ptr as *const u8;
            let bytes = &*(addr as *const [u8;MAX_STR_LEN]);
            from_utf8(&bytes[0..self.len]).expect("Found invalid UTF-8")
        }
    }

    pub fn offset(&mut self, offset: u32) {
        self.bytes_ptr += offset;
    }

    pub fn as_ptr(&mut self) -> *const String {
        self as *const String
    }
}
```

II.V /common/src/syscall.rs

```rust
#[repr(u8)]
pub enum Syscall {
    Puts            = 0x00,
    Putc            = 0x01,
    Exec            = 0x02,
    Keypressed      = 0x03,
    Getc            = 0x04,
    FileStat        = 0x05,
    FileOpen        = 0x06,
    FileClose       = 0x07,
    FileRead        = 0x08,
    FileSeek        = 0x09,
    FileIterator    = 0x0a,
    FileNext        = 0x0b,
    GetTicks        = 0x0c,
    Sleep           = 0x0d,
    SetCursor       = 0x0e,
```

```
    GetCursor       = 0x0f,
    CursorDisable   = 0x10,
    CopyScr         = 0x11,
    AllocFrame      = 0x12,
    FreeFrame       = 0x13
}
```

II.VI /common/src/vga.rs

```rust
pub const BUFFER_HEIGHT : usize = 25;
pub const BUFFER_WIDTH  : usize = 80;
pub const FG_COLOR      : Color = Color::Black;
pub const BG_COLOR      : Color = Color::White;

#[repr(u8)]
#[derive(Copy, Clone)]
pub enum Color {
    Black      = 0x0,
    Blue       = 0x1,
    Green      = 0x2,
    Cyan       = 0x3,
    Red        = 0x4,
    Magenta    = 0x5,
    Brown      = 0x6,
    LightGray  = 0x7,
    DarkGray   = 0x8,
    LightBlue  = 0x9,
    LightGreen = 0xa,
    LightCyan  = 0xb,
    LightRed   = 0xc,
    Pink       = 0xd,
    Yellow     = 0xe,
    White      = 0xf,
}

#[derive(Debug, Clone, Copy)]
pub struct ColorAttribute(u8);

#[derive(Debug, Clone, Copy)]
#[repr(C)]
pub struct Character {
    pub ascii: u8,
    pub attribute: ColorAttribute,
}

pub type FrameBuffer = [[Character; BUFFER_WIDTH]; BUFFER_HEIGHT];

impl Color {
```

```
40      pub fn from_u32(color: u32) -> Color {
41          match color {
42              0x0 => Color::Black,
43              0x1 => Color::Blue,
44              0x2 => Color::Green,
45              0x3 => Color::Cyan,
46              0x4 => Color::Red,
47              0x5 => Color::Magenta,
48              0x6 => Color::Brown,
49              0x7 => Color::LightGray,
50              0x8 => Color::DarkGray,
51              0x9 => Color::LightBlue,
52              0xa => Color::LightGreen,
53              0xb => Color::LightCyan,
54              0xc => Color::LightRed,
55              0xd => Color::Pink,
56              0xe => Color::Yellow,
57              0xf => Color::White,
58              _ => Color::Black
59          }
60      }
61
62      pub fn to_u32(color: Color) -> u32 {
63          match color {
64              Color::Black => 0x0,
65              Color::Blue => 0x1,
66              Color::Green => 0x2,
67              Color::Cyan => 0x3,
68              Color::Red => 0x4,
69              Color::Magenta => 0x5,
70              Color::Brown => 0x6,
71              Color::LightGray => 0x7,
72              Color::DarkGray => 0x8,
73              Color::LightBlue => 0x9,
74              Color::LightGreen => 0xa,
75              Color::LightCyan => 0xb,
76              Color::LightRed => 0xc,
77              Color::Pink => 0xd,
78              Color::Yellow => 0xe,
79              Color::White => 0xf,
80          }
81      }
82  }
83
84  impl ColorAttribute {
85      pub const fn new(background: Color, foreground: Color) ->
    ColorAttribute {
86          ColorAttribute((background as u8) << 4 | (foreground as u8))
```

```
    }
}

impl Character {
    pub const fn null() -> Character {
        Character {
            ascii: 0,
            attribute: ColorAttribute::new(BG_COLOR, FG_COLOR)
        }
    }

    pub const fn new(ascii: u8, attribute: ColorAttribute) -> Character {
        Character {
            ascii: ascii,
            attribute: attribute
        }
    }
}
```

II.VII /common/Cargo.toml

```
[package]
name = "common"
version = "0.1.0"
authors = ["orpheeantoniadis <orphee.antoniadis@gmail.com>"]

[dependencies]
```

III /doc/splash.txt

```
    ____              __  ____  _____
   / __ \__  _______/ /_/ __ \/ ___/
  / /_/ / / / / ___/ __/ / / /\__ \
 / _, _/ /_/ (__  ) /_/ /_/ /___/ /
/_/ |_|\__,_/____/\__/\____//____/

```

IV /grub/menu.lst

```
default=0
timeout=3
title RustOS
kernel /boot/kernel.elf
```

V /kernel

V.I /kernel/src/bootstrap_asm.s

```
1  %include "const.inc"
2
3  extern kernel_start
4  extern kernel_end
5
6  extern low_kernel_start
7  extern low_kernel_end
8
9  extern kmain
10
11 global entrypoint
12 global page_directory
13 global kernel_pt
14
15 ; Values for the multiboot header
16 MULTIBOOT_MAGIC        equ 0x1BADB002
17 MULTIBOOT_ALIGN_MODS   equ 1
18 MULTIBOOT_MEMINFO      equ 2
19 MULTIBOOT_VIDINFO      equ 4
20
21 MULTIBOOT_FLAGS     equ (MULTIBOOT_ALIGN_MODS|MULTIBOOT_MEMINFO)
22
23 ; Magic + checksum + flags must equal 0!
24 MULTIBOOT_CHECKSUM  equ -(MULTIBOOT_MAGIC + MULTIBOOT_FLAGS)
25
26 ;------------------------------------------------------------------------------
27 ; .multiboot section
28 ; This section must be located at the very beginning of the kernel image.
29
30 section .multiboot
31
32 ; Mandatory part of the multiboot header
33 ; see
   http://git.savannah.gnu.org/cgit/grub.git/tree/doc/multiboot.h?h=multiboot
34 dd MULTIBOOT_MAGIC
35 dd MULTIBOOT_FLAGS
36 dd MULTIBOOT_CHECKSUM
37
38 ;------------------------------------------------------------------------------
39 section .low_text
40
41 entrypoint:
42     ; save multiboot infos
43     mov [multiboot_magic], eax
```

```
44      mov [multiboot_info],  ebx
45
46      ; map lower-half kernel pt in pd
47      mov eax, low_kernel_pt
48      mov [page_directory], eax
49      or dword [page_directory], 0x3
50
51      mov eax, 0
52      .low_kernel_pt_init:
53          mov ecx, eax
54          shr ecx, 12
55          and ecx, 0x3ff
56          ; map first MB in lower-half
57          mov [low_kernel_pt + ecx * 4], eax
58          or dword [low_kernel_pt + ecx * 4], 0x3
59          ; map first MB in higher-half
60          mov [kernel_pt + ecx * 4], eax
61          or dword [kernel_pt + ecx * 4], 0x3
62
63          add eax, 0x1000
64          cmp eax, low_kernel_end
65          jl .low_kernel_pt_init
66
67      ; map higher-half kernel pt in pd
68      mov eax, kernel_pt
69      mov [page_directory + KERNEL_PAGE_NUMBER * 4], eax
70      or dword [page_directory + KERNEL_PAGE_NUMBER * 4], 0x3
71
72      mov eax, kernel_start
73      .high_kernel_pt_init:
74          mov ecx, eax
75          shr ecx, 12
76          and ecx, 0x3ff
77
78          mov ebx, eax
79          sub ebx, KERNEL_BASE
80          mov [kernel_pt + ecx * 4], ebx
81          or dword [kernel_pt + ecx * 4], 0x3
82
83          add eax, 0x1000
84          cmp eax, kernel_end
85          jl .high_kernel_pt_init
86
87      ; init paging
88      mov eax, page_directory
89      mov cr3, eax
90      mov eax, cr0
91      or eax, 0x80000000
```

```
92      mov cr0, eax
93
94      lea ecx, [higher_half]
95      jmp ecx
96
97  ;-------------------------------------------------------------------------------
98  section .low_data
99
100 multiboot_magic:
101     dd 0
102 multiboot_info:
103     dd 0
104
105 ;-------------------------------------------------------------------------------
106 section .low_bss nobits
107
108 alignb 4096
109 page_directory:
110     resd 1024
111 low_kernel_pt:
112     resd 1024
113 kernel_pt:
114     resd 1024
115
116 ;-------------------------------------------------------------------------------
117 section .text
118 higher_half:
119     cli  ; disable hardware interrupts
120
121     ; Initialize the stack pointer and EBP (both to the same value)
122     mov esp, stack + STACK_SIZE
123     mov ebp, stack + STACK_SIZE
124
125     ; pass the multiboot infos to the kernel
126     mov ebx, [multiboot_info]
127     add ebx, KERNEL_BASE
128     push ebx
129     push dword [multiboot_magic]
130
131     ; unmap lower-half kernel
132     mov eax, 0
133     mov [page_directory], eax
134
135     call kmain
136
137     .forever:
138         hlt
139         jmp .forever
```

```
140
141 ;-------------------------------------------------------------------------------
142 ; .stack section 1MB long
143 section .stack nobits
144 stack:
145     resb STACK_SIZE ; reserve 1MB for the stack
```

V.II /kernel/src/const.inc

```
1  ; Kernel base address
2  KERNEL_BASE equ 0xC0000000
3  KERNEL_PAGE_NUMBER equ (KERNEL_BASE >> 22)
4
5  ; Kernel stack size
6  STACK_SIZE  equ 0x100000
7
8  ; Must match the values of the same constants in gdt.h!
9  GDT_KERNEL_CODE_SELECTOR  equ     0x08
10 GDT_KERNEL_DATA_SELECTOR  equ     0x10
```

V.III /kernel/src/fs.rs

```
1  #![allow(dead_code)]
2
3  use core::str;
4  use core::mem;
5  use rlibc::memcpy;
6  use ide::*;
7  use vga::*;
8  use common::*;
9
10 const FDT_SIZE : usize = 128;
11 const ENTRY_SIZE : usize = 32;
12 pub const TYPE_TEXT: i32 = 0;
13 pub const TYPE_EXEC: i32 = 1;
14
15 pub static mut FDT: Fdt = [FdtEntry::null();FDT_SIZE];
16 pub static mut SB : Superblock = Superblock::null();
17
18 pub type Fdt = [FdtEntry; FDT_SIZE];
19
20 #[derive(Debug, Clone, Copy)]
21 #[repr(C)]
22 pub struct FdtEntry {
23     pub stat: Stat,
24     pub pos: usize
25 }
```

```
26
27  #[derive(Debug, Clone, Copy)]
28  #[repr(C)]
29  pub struct Superblock {
30      pub block_size: usize,
31      pub fat_size: usize,
32      pub root_entry: usize
33  }
34
35  pub trait StatBuilder {
36      fn new(filename: &str) -> Self;
37  }
38
39  pub trait FileIteratorBuilder {
40      fn new() -> Self;
41      fn has_next(&mut self) -> bool;
42      fn next(&mut self, filename: *mut u8) -> i8;
43  }
44
45  pub fn file_exists(filename: &str) -> bool {
46      let mut raw_filename = [0;MAX_FILENAME_LENGTH];
47      let mut it = FileIterator::new();
48      while it.has_next() {
49          it.next(&mut raw_filename[0]);
50          let it_filename = bytes_to_str(&raw_filename);
51          if it_filename == filename {
52              return true;
53          }
54      }
55      return false;
56  }
57
58  pub fn file_open(filename: &str) -> i32 {
59      unsafe {
60          if file_exists(filename) {
61              let fd = free_fd();
62              FDT[fd as usize].stat = Stat::new(filename);
63              return fd;
64          }
65          return -1;
66      }
67  }
68
69  pub fn file_read(fd: i32, buf: *mut u8, n: usize) -> i32 {
70      unsafe {
71          if fd < 0 || FDT[fd as usize].stat.start == 0 {
72              return -1;
73          }
```

```
74
75          let mut sector : [u16;SECTOR_SIZE/2] = [0;SECTOR_SIZE/2];
76          read_sector(1, &mut sector[0] as *mut u16);
77          let fat = mem::transmute::<[u16;SECTOR_SIZE/2],
    [u8;SECTOR_SIZE]>(sector);
78
79          let mut cnt = 0;
80          let mut block = FDT[fd as usize].stat.start;
81          let mut sector_id = block * (SB.block_size / SECTOR_SIZE);
82          let size = if FDT[fd as usize].pos + n > FDT[fd as
    usize].stat.size {
83              FDT[fd as usize].stat.size
84          } else {
85              FDT[fd as usize].pos + n
86          };
87
88          for i in 0..(size / SECTOR_SIZE) {
89              if i >= FDT[fd as usize].pos / SECTOR_SIZE {
90                  sector_id += i % (SB.block_size / SECTOR_SIZE);
91                 read_sector(sector_id as u32, &mut sector[0] as *mut u16);
92                  let data = mem::transmute::<[u16;SECTOR_SIZE/2],
    [u8;SECTOR_SIZE]>(sector);
93                  memcpy(buf.offset(cnt as isize), &data[0], SECTOR_SIZE);
94                  FDT[fd as usize].pos += SECTOR_SIZE;
95                  cnt += SECTOR_SIZE;
96              }
97              if FDT[fd as usize].pos % SB.block_size == 0 {
98                  block = fat[block] as usize;
99                  sector_id = block * (SB.block_size / SECTOR_SIZE);
100             }
101         }
102
103         if FDT[fd as usize].pos >= FDT[fd as usize].stat.size {
104             return 0;
105         } else {
106             read_sector(sector_id as u32, &mut sector[0] as *mut u16);
107             let data = mem::transmute::<[u16;SECTOR_SIZE/2],
    [u8;SECTOR_SIZE]>(sector);
108            memcpy(buf.offset(cnt as isize), &data[FDT[fd as usize].pos %
    SECTOR_SIZE], size % SECTOR_SIZE);
109             FDT[fd as usize].pos += size % SECTOR_SIZE;
110             return n as i32;
111         }
112     }
113 }
114
115 pub fn file_seek(fd: i32, offset: usize) -> i32 {
116     unsafe {
```

```
117         if FDT[fd as usize].pos + offset > FDT[fd as usize].stat.size {
118             FDT[fd as usize].pos = FDT[fd as usize].stat.size;
119             return -1;
120         } else {
121             FDT[fd as usize].pos += offset;
122             return 0;
123         }
124     }
125 }
126 
127 pub fn rewind(fd: i32) {
128     unsafe {
129         FDT[fd as usize].pos = 0;
130     }
131 }
132 
133 pub fn file_close(fd: i32) -> i32 {
134     if fd < 0 || unsafe { FDT[fd as usize].stat.start } == 0 {
135         println!("fd {} does not exist.", fd);
136         return -1;
137     } else {
138         unsafe { FDT[fd as usize] = FdtEntry::null() };
139         return 0;
140     }
141 }
142 
143 pub fn file_type(fd: i32) -> i32 {
144     let mut buf = [0;MAX_STR_LEN];
145     if file_read(fd, &mut buf[0], MAX_STR_LEN) != -1 {
146         rewind(fd);
147         if bytes_to_str(&buf) != "\0" {
148             return TYPE_TEXT;
149         } else {
150             return TYPE_EXEC;
151         }
152     }
153     return -1;
154 }
155 
156 pub fn set_superblock() {
157     unsafe { SB = Superblock::new(); }
158 }
159 
160 fn free_fd() -> i32 {
161     unsafe {
162         let mut cnt = 0;
163         for entry in FDT.iter() {
164             if entry.stat.start == 0 {
```

```
                return cnt;
            }
            cnt += 1;
        }
        return -1;
    }
}

impl FdtEntry {
    const fn null() -> FdtEntry {
        FdtEntry {
            stat: Stat {
                name: [0;MAX_FILENAME_LENGTH],
                size: 0,
                entry_offset: 0,
                start: 0
            },
            pos: 0
        }
    }
}

impl StatBuilder for Stat {
    fn new(filename: &str) -> Stat {
        let mut sector : [u16;SECTOR_SIZE/2] = [0;SECTOR_SIZE/2];
        let mut raw_filename = [0;MAX_FILENAME_LENGTH];
        let mut it = FileIterator::new();
        while it.has_next() {
            it.next(&mut raw_filename[0]);
            if filename == bytes_to_str(&raw_filename) {
                read_sector(it.sector, &mut sector[0] as *mut u16);
                unsafe {
                    let offset = it.offset - ENTRY_SIZE;
                    let entries = mem::transmute::<[u16;SECTOR_SIZE/2],
[u8;SECTOR_SIZE]>(sector);
                    let start = [entries[offset+26], entries[offset+27]];
                    let start = mem::transmute::<[u8;2], u16>(start);
                    let size = [entries[offset+28], entries[offset+29],
entries[offset+30], entries[offset+31]];
                    let size = mem::transmute::<[u8;4], u32>(size);
                    return Stat {
                        name: raw_filename,
                        size: size as usize,
                        entry_offset: offset as u16,
                        start: start as usize
                    }
                }
            }
```

```
211         }
212         Stat { name: raw_filename, size: 0, entry_offset: 0, start: 0 }
213     }
214 }
215 
216 impl FileIteratorBuilder for FileIterator {
217     fn new() -> FileIterator {
218         FileIterator {
219             sector: (unsafe { SB.root_entry * SB.block_size } /
SECTOR_SIZE) as u32,
220             offset: 0
221         }
222     }
223 
224     fn has_next(&mut self) -> bool {
225         if self.sector < self.sector + unsafe { SB.block_size /
SECTOR_SIZE } as u32 {
226             let mut sector : [u16;SECTOR_SIZE/2] = [0;SECTOR_SIZE/2];
227             read_sector(self.sector, &mut sector[0] as *mut u16);
228             let entries = unsafe {
229                 mem::transmute::<[u16;SECTOR_SIZE/2],
[u8;SECTOR_SIZE]>(sector)
230             };
231             if entries[self.offset] != 0 {
232                 return true;
233             }
234         }
235         return false;
236     }
237 
238     fn next(&mut self, filename: *mut u8) -> i8 {
239         unsafe {
240             if self.has_next() {
241                 let mut sector : [u16;SECTOR_SIZE/2] = [0;SECTOR_SIZE/2];
242                 read_sector(self.sector, &mut sector[0] as *mut u16);
243                 let entries = mem::transmute::<[u16;SECTOR_SIZE/2],
[u8;SECTOR_SIZE]>(sector);
244                 memcpy(filename, &entries[self.offset],
MAX_FILENAME_LENGTH);
245                 self.offset = (self.offset + ENTRY_SIZE) % SECTOR_SIZE;
246                 if self.offset == 0 {
247                     self.sector += 1;
248                 }
249                 return 0;
250             }
251             return -1;
252         }
253     }
```

```
}

impl Superblock {
    const fn null() -> Superblock {
        Superblock { block_size: 0, fat_size: 0, root_entry: 0 }
    }

    fn new() -> Superblock {
        let mut sector : [u16;SECTOR_SIZE/2] = [0;SECTOR_SIZE/2];
        read_sector(0, &mut sector[0] as *mut u16);
        let raw_sb = unsafe {
            mem::transmute::<[u16;SECTOR_SIZE/2],
[u8;SECTOR_SIZE]>(sector)
        };
        let label = bytes_to_str(&raw_sb[0x52..0x59]);
        let block_size = raw_sb[13] as usize * SECTOR_SIZE;
        let fat_size = unsafe {
            mem::transmute::<[u8;4], u32>([raw_sb[0x24], raw_sb[0x25],
raw_sb[0x26], raw_sb[0x27]])
        };
        let root_entry = raw_sb[0x2c];
        println!("\n{} ready.", label);
        println!("Block size = {} bytes", block_size);
        println!("FAT size = {} bytes", fat_size);
        println!("Root entry = block number {}\n", root_entry);

        Superblock {
            block_size: block_size,
            fat_size: fat_size as usize,
            root_entry: root_entry as usize
        }
    }
}
```

V.IV /kernel/src/gdt_asm.s

```
%include "const.inc"

global gdt_load

section .text:           ; start of the text (code) section

gdt_load:
    mov     eax,[esp+4]  ; Get the pointer to the GDT, passed as a
parameter.
    lgdt    [eax]        ; Load the new GDT pointer
    mov     ax,GDT_KERNEL_DATA_SELECTOR   ; offset in the GDT of the
kernel data segment
```

```
    mov     ds,ax         ; Load all data segment selectors
    mov     es,ax
    mov     fs,ax
    mov     gs,ax
    mov     ss,ax
    jmp     GDT_KERNEL_CODE_SELECTOR:.flush  ; far jump [selector:offset]
    .flush:
        ret
```

V.V /kernel/src/gdt.rs

```
//! Module for the memory management of RustOS using a Global
Descriptor Table

#![allow(dead_code)]

use core::mem::size_of;
use x86::*;
use task::*;

/// The GDT size (not including the tss and ldt entries)
pub const GDT_SIZE: usize = 6;

/// Converts a descriptor index in the GDT into a selector
pub const fn gdt_index_to_selector(idx: u32) -> u32 {idx << 3}
/// Converts a descriptor selector in the GDT into an index
pub const fn selector_to_gdt_index(idx: u32) -> u32 {idx >> 3}

/// The Global Descriptor Table of RustOS
pub static mut GDT: Gdt = [GdtEntry::null();GDT_SIZE+TASKS_NB];
static mut GDT_PTR: GdtPtr = GdtPtr::null();

/// Defines a Global Descriptor Table
pub type Gdt = [GdtEntry; GDT_SIZE+TASKS_NB];

/// Structure of a GDT descriptor. There are 2 types of descriptors:
segments and TSS.
/// Section 3.4.5 of Intel 64 & IA32 architectures software developer's
manual describes
/// segment descriptors while section 6.2.2 describes TSS descriptors.
#[derive(Debug, Clone, Copy)]
#[repr(C, packed)]
pub struct GdtEntry {
    lim15_0: u16,
    base15_0: u16,
    base23_16: u8,

    flags7_0: u8,
```

```
35      flags15_8: u8,
36
37      base31_24: u8
38  }
39
40  // Structure describing a pointer to the GDT descriptor table.
41  // This format is required by the lgdt instruction.
42  #[derive(Debug, Clone, Copy)]
43  #[repr(C, packed)]
44  struct GdtPtr {
45      limit: u16, // Limit of the table (ie. its size)
46      base: *const Gdt // Address of the first entry
47  }
48
49  extern "C" {
50      fn gdt_load(gdt_ptr: *const GdtPtr);
51  }
52
53  /// Initialize the Global Descriptor Table
54  pub fn gdt_init() {
55      unsafe {
56          // initialize 3 segment descriptors: NULL, code segment, data
    segment.
57          // Code and data segments must have a privilege level of 0.
58          GDT[0] = GdtEntry::null();
59          GDT[1] = GdtEntry::make_code_segment(0, 0xfffff, DPL_KERNEL);
60          GDT[2] = GdtEntry::make_data_segment(0, 0xfffff, DPL_KERNEL);
61          GDT[3] = GdtEntry::make_code_segment(0, 0xfffff, DPL_USER);
62          GDT[4] = GdtEntry::make_data_segment(0, 0xfffff, DPL_USER);
63          // setup gdt_ptr so it points to the GDT and ensure it has the
    right limit.
64          GDT_PTR = GdtPtr::new((size_of::<Gdt>() - 1) as u16, &GDT);
65          // Load the GDT
66          gdt_load(&GDT_PTR);
67          // Init tasks
68          tasks_init();
69      }
70  }
71
72  impl GdtEntry {
73      /// Create a null segment
74      pub const fn null() -> GdtEntry {
75          GdtEntry {
76              lim15_0:    0,
77              base15_0:   0,
78              base23_16:  0,
79              flags7_0:   0,
80              flags15_8:  0,
```

```
81              base31_24:  0
82          }
83      }
84
85      fn build_entry(base: u32, limit: u32, gdt_type: u8, s: u8, db: u8,
    granularity: u8, dpl: u8) -> GdtEntry {
86          GdtEntry {
87              lim15_0:    (limit & 0xffff) as u16,
88              base15_0:   (base & 0xffff) as u16,
89              base23_16:  ((base >> 16) & 0xff) as u8,
90              flags7_0:   gdt_type | s<<4 | dpl<<5 | 1<<7,
91              flags15_8:  ((limit >> 16) & 0xf) as u8 | db<<6 |
    granularity<<7,
92              base31_24:  ((base >> 24) & 0xff) as u8
93          }
94      }
95
96      /// Create a code segment specified by the base, limit and privilege
    level passed in arguments
97      pub fn make_code_segment(base: u32, limit: u32, dpl: u8) -> GdtEntry
    {
98          GdtEntry::build_entry(base, limit, TYPE_CODE_EXECREAD,
    S_CODE_OR_DATA, DB_SEG, 1, dpl)
99      }
100
101     /// Create a data segment specified by the base, limit and privilege
    level passed in arguments
102     pub fn make_data_segment(base: u32, limit: u32, dpl: u8) -> GdtEntry
    {
103         GdtEntry::build_entry(base, limit, TYPE_DATA_READWRITE,
    S_CODE_OR_DATA, DB_SEG, 1, dpl)
104     }
105
106     /// Create a TSS segment
107     pub fn make_tss(base: u32, dpl: u8) -> GdtEntry {
108         GdtEntry::build_entry(base, (size_of::<Tss>() - 1) as u32,
    TYPE_TSS, S_SYSTEM, DB_SYS, 0, dpl)
109     }
110
111     /// Create an LDT segment
112     pub fn make_ldt(base: u32, limit: u32, dpl: u8) -> GdtEntry {
113         GdtEntry::build_entry(base, limit, TYPE_LDT, S_SYSTEM, DB_SYS, 0,
    dpl)
114     }
115
116     /// Converts a GDT entry to its index in the GDT
117     pub fn to_index(&mut self) -> u32 {
118         unsafe {
```

```
            ((self as *mut _ as u32) - (&GDT as *const _ as u32)) >> 3
        }
    }

    /// Converts a GDT entry to its selector in the GDT
    pub fn to_selector(&mut self) -> u32 {
        unsafe {
            (self as *mut _ as u32) - (&GDT as *const _ as u32)
        }
    }
}

impl GdtPtr {
    const fn null() -> GdtPtr {
        GdtPtr {
            limit:  0,
            base:   0 as *const _
        }
    }

    fn new(limit: u16, base: *const Gdt) -> GdtPtr {
        GdtPtr {
            limit:  limit,
            base:   base
        }
    }
}
```

V.VI /kernel/src/ide.rs

```
#![allow(dead_code)]

/**
* Simple IDE read/write routines using PIO mode.
* This code is very CPU intensive and not efficient
* (if that's what you're after, use DMA mode instead).
* Reference: http://wiki.osdev.org/ATA_PIO_Mode
* ATA disk0, I/O ports: 0x1f0-0x1f7, 0x3f6
* ATA disk1, I/O ports: 0x170-0x177, 0x376
*/

use pio::*;

// IDE ports
const IDE_CMD : u16 = 0x1f7;
const IDE_DATA : u16 = 0x1f0;

pub const SECTOR_SIZE : usize = 512;
```

```
19
20  /**
21   * Wait for the disk drive to be ready.
22   */
23  fn wait_drive() {
24      unsafe { while(inb(IDE_CMD) & 192) != 64 { } }
25  }
26
27  /**
28   * Prepare the disk drive for read/write at the specified sector in LBA
    mode.
29   * @param sector the sector to read or write (0-indexed).
30   */
31  fn pio_prepare(sector: u32) {
32      unsafe {
33          wait_drive();
34          outb(0x1f2, 1);                                        // 1 sector
35          outb(0x1f3, (sector & 0xff) as u8);                      // send
    bits 0-7 of LBA
36          outb(0x1f4, ((sector >> 8) & 0xff) as u8);               // send
    bits 8-15 of LBA
37          outb(0x1f5, ((sector >> 16) & 0xff) as u8);              // send
    bits 16-23 of LBA
38          outb(0x1f6, ((sector >> 24) & 0x0f) as u8 | 0xe0);      // send
    bits 24-27 of LBA + set LBA mode; 0xe0 = 11100000b;
39      }
40  }
41
42  /**
43   * Read sectors from the first disk.
44   * @param sector first sector to read (0-indexed)
45   * @param dst address to store to read data
46   * Based on the assembly code at
    http://wiki.osdev.org/ATA_read/write_sectors
47   */
48  pub fn read_sector(sector: u32, dst: *mut u16) {
49      unsafe {
50          pio_prepare(sector);
51
52          outb(IDE_CMD, 0x20);  // read with retry
53          wait_drive();
54
55          for i in 0..(SECTOR_SIZE/2) {
56              *dst.offset(i as isize) = inw(IDE_DATA);
57          }
58      }
59  }
60
```

```rust
/**
 * Write sectors from the first disk.
 * @param sector first sector to write (0-indexed)
 * @param src address of the data to be written
 */
pub fn write_sector(sector: u32, src: *mut u16) {
    unsafe {
        pio_prepare(sector);

        outb(IDE_CMD, 0x30);  // write with retry
        wait_drive();

        for i in 0..(SECTOR_SIZE/2) {
            outw(IDE_DATA, *src.offset(i as isize));
        }
    }
}
```

V.VII /kernel/src/idt_asm.s

```
%include "const.inc"

extern exception_handler
extern irq_handler

section .text   ; start of the text (code) section
align 4         ; the code must be 4 byte aligned

;---------------------------------------------------
; CPU exceptions
; Macro to generate exceptions. The only argument is for exception's
digit
%macro exception 1
global _exception_%1
_exception_%1:
  cli             ; disable interrupts
    ; this if is for exceptions without error code
    %if %1 < 8 || %1 == 9 || %1 == 15 || %1 == 16 || %1 > 17
    push     0          ; dummy error code in certain case
    %endif
    push     %1        ; exception number
    jmp     exception_wrapper
%endmacro
; Creation of all exceptions (0 to 20), total = 21
%assign i 0
%rep 21
exception i
%assign i i+1
```

```
28 %endrep
29
30 ;-----------------------------------------------
31 ; IRQ
32 ; Macro for irq
33 %macro irq 1
34 global _irq_%1
35 _irq_%1:
36     cli      ; disable interrupts
37     push 0   ; dummy error code
38     push %1  ; irq number
39     jmp irq_wrapper
40 %endmacro
41 ; Creation of all irq (0 to 15), total = 16
42 %assign i 0
43 %rep 16
44 irq i
45 %assign i i+1
46 %endrep
47
48 ;-----------------------------------------------
49 ; Wrapper for exceptions
50 %macro wrapper 1
51 %1_wrapper:
52 ; Save all registers
53     push    eax
54     push    ebx
55     push    ecx
56     push    edx
57     push    esi
58     push    edi
59     push    ebp
60     push    ds
61     push    es
62     push    fs
63     push    gs
64
65     ; Load kernel data descriptor into all segments
66     mov     ax,GDT_KERNEL_DATA_SELECTOR
67     mov     ds,ax
68     mov     es,ax
69     mov     fs,ax
70     mov     gs,ax
71
72     ; Pass the stack pointer (which gives the CPU context) to the C
   function
73     mov     eax,esp
74     push    eax
```

```
    call    %1_handler  ; implemented in idt.c
    pop     eax  ; only here to balance the "push eax" done before the
call

    ; Restore all registers
    pop     gs
    pop     fs
    pop     es
    pop     ds
    pop     ebp
    pop     edi
    pop     esi
    pop     edx
    pop     ecx
    pop     ebx
    pop     eax

    ; Fix the stack pointer due to the 2 push done before the call to
    ; exception_wrapper: error code and exception/irq number
    add     esp,8
    iret
%endmacro

wrapper exception
wrapper irq

;-----------------------------------------------
; Load the IDT
global idt_load
; Argument : address of idt structure
idt_load:
  mov eax, [esp + 4]
  lidt [eax]
  ret
```

V.VIII /kernel/src/idt.rs

```
//! Module for the intrruption management of RustOS using an Interrupt
Descriptor Table
#![allow(dead_code)]

use core::mem::size_of;
use x86::*;
use vga::*;
use pic::*;
use timer::timer_handler;
use keyboard::keyboard_handler;
use syscall::_syscall_handler;
```

```
11
12  const IDT_SIZE: usize = 256;
13  const EXCEPTION_MESSAGES: [&str;21] = [
14      "EXCEPTION 0 : Divide error",
15      "EXCEPTION 1 : Intel RESERVED exception number",
16      "EXCEPTION 2 : External non maskable interrupt",
17      "EXCEPTION 3 : Breakpoint",
18      "EXCEPTION 4 : Overflow",
19      "EXCEPTION 5 : Bound range exceeded",
20      "EXCEPTION 6 : Invalid opcode",
21      "EXCEPTION 7 : Device not available",
22      "EXCEPTION 8 : Double fault",
23      "EXCEPTION 9 : Coprocessor segment overrun",
24      "EXCEPTION 10 : Invalid TSS",
25      "EXCEPTION 11 : Segment not present",
26      "EXCEPTION 12 : Stack-segment fault",
27      "EXCEPTION 13 : General protection",
28      "EXCEPTION 14 : Page fault",
29      "EXCEPTION 15 : Intel RESERVED exception number",
30      "EXCEPTION 16 : x87 FPU floating-point error",
31      "EXCEPTION 17 : Alignment check",
32      "EXCEPTION 18 : Machine check",
33      "EXCEPTION 19 : SIMD floating-point exception",
34      "EXCEPTION 20 : Virtualization exception"
35  ];
36
37  static mut IDT: Idt = [IdtEntry::null();IDT_SIZE];
38  static mut IDT_PTR: IdtPtr = IdtPtr::null();
39
40  type Idt = [IdtEntry; IDT_SIZE];
41
42  // Structure of an IDT descriptor. There are 3 types of descriptors:
43  // a task-gate, an interrupt-gate, and a trap-gate.
44  // See 5.11 of Intel 64 & IA32 architectures software developer's
    manual for more details.
45  // For task gates, offset must be 0.
46  #[derive(Debug, Clone, Copy)]
47  #[repr(C, packed)]
48  struct IdtEntry {
49      offset15_0: u16,   // only used by trap and interrupt gates
50      selector: u16,     // segment selector for trap and interrupt gates;
    TSS segment selector for task gates
51      reserved: u8,
52      flags: u8,
53      offset31_16: u16   // only used by trap and interrupt gates
54  }
55
56  // Structure describing a pointer to the IDT gate table.
```

```rust
// This format is required by the lgdt instruction.
#[derive(Debug, Clone, Copy)]
#[repr(C, packed)]
struct IdtPtr {
    limit: u16, // Limit of the table (ie. its size)
    base: *const Idt   // Address of the first entry
}

/// CPU context used when saving/restoring context from an interrupt
#[derive(Debug, Clone, Copy)]
#[repr(C, packed)]
pub struct Regs {
    gs: u32, fs: u32, es: u32, ds: u32,
    ebp: u32, edi: u32, esi: u32,
    edx: u32, ecx: u32, ebx: u32, eax: u32,
    number: u32, error_code: u32,
    eip: u32, cs: u32, eflags: u32, esp: u32, ss: u32
}

/// Initialize the Interrupt Descriptor Table
pub fn idt_init() {
    unsafe {
        // CPU exceptions
        IDT[0] = IdtEntry::new(GDT_KERNEL_CODE_SELECTOR as u16, _exception_0 as *const () as u32, TYPE_INTERRUPT_GATE, DPL_KERNEL);
        IDT[1] = IdtEntry::new(GDT_KERNEL_CODE_SELECTOR as u16, _exception_1 as *const () as u32, TYPE_INTERRUPT_GATE, DPL_KERNEL);
        IDT[2] = IdtEntry::new(GDT_KERNEL_CODE_SELECTOR as u16, _exception_2 as *const () as u32, TYPE_INTERRUPT_GATE, DPL_KERNEL);
        IDT[3] = IdtEntry::new(GDT_KERNEL_CODE_SELECTOR as u16, _exception_3 as *const () as u32, TYPE_INTERRUPT_GATE, DPL_KERNEL);
        IDT[4] = IdtEntry::new(GDT_KERNEL_CODE_SELECTOR as u16, _exception_4 as *const () as u32, TYPE_INTERRUPT_GATE, DPL_KERNEL);
        IDT[5] = IdtEntry::new(GDT_KERNEL_CODE_SELECTOR as u16, _exception_5 as *const () as u32, TYPE_INTERRUPT_GATE, DPL_KERNEL);
        IDT[6] = IdtEntry::new(GDT_KERNEL_CODE_SELECTOR as u16, _exception_6 as *const () as u32, TYPE_INTERRUPT_GATE, DPL_KERNEL);
        IDT[7] = IdtEntry::new(GDT_KERNEL_CODE_SELECTOR as u16, _exception_7 as *const () as u32, TYPE_INTERRUPT_GATE, DPL_KERNEL);
        IDT[8] = IdtEntry::new(GDT_KERNEL_CODE_SELECTOR as u16, _exception_8 as *const () as u32, TYPE_INTERRUPT_GATE, DPL_KERNEL);
        IDT[9] = IdtEntry::new(GDT_KERNEL_CODE_SELECTOR as u16, _exception_9 as *const () as u32, TYPE_INTERRUPT_GATE, DPL_KERNEL);
        IDT[10] = IdtEntry::new(GDT_KERNEL_CODE_SELECTOR as u16, _exception_10 as *const () as u32, TYPE_INTERRUPT_GATE, DPL_KERNEL);
        IDT[11] = IdtEntry::new(GDT_KERNEL_CODE_SELECTOR as u16, _exception_11 as *const () as u32, TYPE_INTERRUPT_GATE, DPL_KERNEL);
```

```
92          IDT[12] = IdtEntry::new(GDT_KERNEL_CODE_SELECTOR as u16,
    _exception_12 as *const () as u32, TYPE_INTERRUPT_GATE, DPL_KERNEL);
93          IDT[13] = IdtEntry::new(GDT_KERNEL_CODE_SELECTOR as u16,
    _exception_13 as *const () as u32, TYPE_INTERRUPT_GATE, DPL_KERNEL);
94          IDT[14] = IdtEntry::new(GDT_KERNEL_CODE_SELECTOR as u16,
    _exception_14 as *const () as u32, TYPE_INTERRUPT_GATE, DPL_KERNEL);
95          IDT[15] = IdtEntry::new(GDT_KERNEL_CODE_SELECTOR as u16,
    _exception_15 as *const () as u32, TYPE_INTERRUPT_GATE, DPL_KERNEL);
96          IDT[16] = IdtEntry::new(GDT_KERNEL_CODE_SELECTOR as u16,
    _exception_16 as *const () as u32, TYPE_INTERRUPT_GATE, DPL_KERNEL);
97          IDT[17] = IdtEntry::new(GDT_KERNEL_CODE_SELECTOR as u16,
    _exception_17 as *const () as u32, TYPE_INTERRUPT_GATE, DPL_KERNEL);
98          IDT[18] = IdtEntry::new(GDT_KERNEL_CODE_SELECTOR as u16,
    _exception_18 as *const () as u32, TYPE_INTERRUPT_GATE, DPL_KERNEL);
99          IDT[19] = IdtEntry::new(GDT_KERNEL_CODE_SELECTOR as u16,
    _exception_19 as *const () as u32, TYPE_INTERRUPT_GATE, DPL_KERNEL);
100         IDT[20] = IdtEntry::new(GDT_KERNEL_CODE_SELECTOR as u16,
    _exception_20 as *const () as u32, TYPE_INTERRUPT_GATE, DPL_KERNEL);
101
102         // IRQ
103         IDT[32] = IdtEntry::new(GDT_KERNEL_CODE_SELECTOR as u16, _irq_0
    as *const () as u32, TYPE_INTERRUPT_GATE, DPL_KERNEL);
104         IDT[33] = IdtEntry::new(GDT_KERNEL_CODE_SELECTOR as u16, _irq_1
    as *const () as u32, TYPE_INTERRUPT_GATE, DPL_KERNEL);
105         IDT[34] = IdtEntry::new(GDT_KERNEL_CODE_SELECTOR as u16, _irq_2
    as *const () as u32, TYPE_INTERRUPT_GATE, DPL_KERNEL);
106         IDT[35] = IdtEntry::new(GDT_KERNEL_CODE_SELECTOR as u16, _irq_3
    as *const () as u32, TYPE_INTERRUPT_GATE, DPL_KERNEL);
107         IDT[36] = IdtEntry::new(GDT_KERNEL_CODE_SELECTOR as u16, _irq_4
    as *const () as u32, TYPE_INTERRUPT_GATE, DPL_KERNEL);
108         IDT[37] = IdtEntry::new(GDT_KERNEL_CODE_SELECTOR as u16, _irq_5
    as *const () as u32, TYPE_INTERRUPT_GATE, DPL_KERNEL);
109         IDT[38] = IdtEntry::new(GDT_KERNEL_CODE_SELECTOR as u16, _irq_6
    as *const () as u32, TYPE_INTERRUPT_GATE, DPL_KERNEL);
110         IDT[39] = IdtEntry::new(GDT_KERNEL_CODE_SELECTOR as u16, _irq_7
    as *const () as u32, TYPE_INTERRUPT_GATE, DPL_KERNEL);
111         IDT[40] = IdtEntry::new(GDT_KERNEL_CODE_SELECTOR as u16, _irq_8
    as *const () as u32, TYPE_INTERRUPT_GATE, DPL_KERNEL);
112         IDT[41] = IdtEntry::new(GDT_KERNEL_CODE_SELECTOR as u16, _irq_9
    as *const () as u32, TYPE_INTERRUPT_GATE, DPL_KERNEL);
113         IDT[42] = IdtEntry::new(GDT_KERNEL_CODE_SELECTOR as u16, _irq_10
    as *const () as u32, TYPE_INTERRUPT_GATE, DPL_KERNEL);
114         IDT[43] = IdtEntry::new(GDT_KERNEL_CODE_SELECTOR as u16, _irq_11
    as *const () as u32, TYPE_INTERRUPT_GATE, DPL_KERNEL);
115         IDT[44] = IdtEntry::new(GDT_KERNEL_CODE_SELECTOR as u16, _irq_12
    as *const () as u32, TYPE_INTERRUPT_GATE, DPL_KERNEL);
116         IDT[45] = IdtEntry::new(GDT_KERNEL_CODE_SELECTOR as u16, _irq_13
    as *const () as u32, TYPE_INTERRUPT_GATE, DPL_KERNEL);
```

```
117         IDT[46] = IdtEntry::new(GDT_KERNEL_CODE_SELECTOR as u16, _irq_14
    as *const () as u32, TYPE_INTERRUPT_GATE, DPL_KERNEL);
118         IDT[47] = IdtEntry::new(GDT_KERNEL_CODE_SELECTOR as u16, _irq_15
    as *const () as u32, TYPE_INTERRUPT_GATE, DPL_KERNEL);
119
120         // Syscall
121         IDT[48] = IdtEntry::new(GDT_KERNEL_CODE_SELECTOR as u16,
    _syscall_handler as *const () as u32, TYPE_TRAP_GATE, DPL_USER);
122
123         // setup idt_ptr so it points to the IDT and ensure it has the
    right limit.
124         IDT_PTR = IdtPtr::new((size_of::<Idt>() - 1) as u16, &IDT);
125         // Load the IDT
126         idt_load(&IDT_PTR);
127     }
128 }
129
130 /// Handler called by the low-level subroutine exception_wrapper
    contain in idt_asm.s
131 /// when an exception occurs
132 #[no_mangle]
133 pub extern fn exception_handler(regs: *mut Regs) {
134     unsafe {
135         panic!(EXCEPTION_MESSAGES[(*regs).number as usize]);
136     }
137 }
138
139 /// Handler called by the low-level subroutine irq_wrapper contain in
    idt_asm.s
140 /// when an interruption occurs
141 #[no_mangle]
142 pub extern fn irq_handler(regs: *mut Regs) {
143     let irq = unsafe { (*regs).number };
144     match irq {
145         0 => timer_handler(),
146         1 => keyboard_handler(),
147         14 => (),
148         _ => println!("irq {} not implemented", irq)
149     }
150     pic_eoi(irq);
151 }
152
153 impl IdtEntry {
154     const fn null() -> IdtEntry {
155         IdtEntry {
156             offset15_0: 0,
157             selector: 0,
158             reserved: 0,
```

```
159             flags: 0,
160             offset31_16: 0
161         }
162     }
163 
164     // Build and return an IDT entry.
165     // selector is the code segment selector to access the ISR
166     // offset is the address of the ISR (for task gates, offset must be
    0)
167     // type indicates the IDT entry type
168     // dpl is the privilege level required to call the associated ISR
169     fn new(selector: u16, offset: u32, idt_type: u8, dpl: u8) ->
    IdtEntry {
170         IdtEntry {
171             offset15_0: (offset & 0xffff) as u16,
172             selector: selector,
173             reserved: 0,
174             flags: idt_type | dpl<<5 | 1<<7,
175             offset31_16: ((offset >> 16) & 0xffff) as u16
176         }
177     }
178 }
179 
180 impl IdtPtr {
181     const fn null() -> IdtPtr {
182         IdtPtr {
183             limit: 0,
184             base: 0 as *const _
185         }
186     }
187 
188     fn new(limit: u16, base: *const Idt) -> IdtPtr {
189         IdtPtr {
190             limit:  limit,
191             base:   base
192         }
193     }
194 }
195 
196 extern "C" {
197     fn idt_load(idt_ptr: *const IdtPtr);
198 
199     // Exception handler
200     fn _exception_0();
201     fn _exception_1();
202     fn _exception_2();
203     fn _exception_3();
204     fn _exception_4();
```

```
205     fn _exception_5();
206     fn _exception_6();
207     fn _exception_7();
208     fn _exception_8();
209     fn _exception_9();
210     fn _exception_10();
211     fn _exception_11();
212     fn _exception_12();
213     fn _exception_13();
214     fn _exception_14();
215     fn _exception_15();
216     fn _exception_16();
217     fn _exception_17();
218     fn _exception_18();
219     fn _exception_19();
220     fn _exception_20();
221 
222     // Interruption handler
223     fn _irq_0();
224     fn _irq_1();
225     fn _irq_2();
226     fn _irq_3();
227     fn _irq_4();
228     fn _irq_5();
229     fn _irq_6();
230     fn _irq_7();
231     fn _irq_8();
232     fn _irq_9();
233     fn _irq_10();
234     fn _irq_11();
235     fn _irq_12();
236     fn _irq_13();
237     fn _irq_14();
238     fn _irq_15();
239 }
```

V.IX /kernel/src/kernel.ld

```
1  /* the entry point */
2  ENTRY(entrypoint)
3  
4  SECTIONS {
5      /* Low memory Kernel */
6      . = 0x00100000;
7      low_kernel_start = .;
8      .boot ALIGN(4) :
9      {
10         *(.multiboot)
```

```
    }
    .low_text ALIGN (4K) :
    {
        *(.low_text)
    }
    .low_data ALIGN (4K) :
    {
        *(.low_data)
    }
    .low_bss ALIGN (4K) :
    {
        low_bss_start = .;
        *(.low_bss)
        low_bss_end = .;
    }
    low_kernel_end = .;

    /* Higher-half Kernel */
    . += 0xC0000000;
    kernel_start = .;
    /* kernel stack */
    .stack ALIGN(4) : AT(ADDR(.stack) - 0xC0000000)
    {
        *(.stack)
    }
    /* code */
    .text ALIGN(4K) : AT(ADDR(.text) - 0xC0000000)
    {
        *(.text*)
    }
    /* read-only data */
    .rodata ALIGN(4K) : AT(ADDR(.rodata) - 0xC0000000)
    {
        *(.rodata*)
    }
    /* initialized data */
    .data ALIGN(4K) : AT(ADDR(.data) - 0xC0000000)
    {
        *(.data*)
    }
    /* unitialized data */
    .bss ALIGN(4K) : AT(ADDR(.bss) - 0xC0000000)
    {
        bss_start = .;
        *(COMMON)
        *(.bss*)
        bss_end = .;
    }
```

```
    kernel_end = .;

    /* Utils */
    low_kernel_size = low_kernel_end - low_kernel_start;
    low_bss_size = low_bss_end - low_bss_start;
    kernel_size = kernel_end - kernel_start;
    bss_size = bss_end - bss_start;
}

ASSERT(kernel_size < 0x300000, "Kernel exceeds the 3 MB limit!");
```

V.X /kernel/src/kernel.rs

```
//! # RustOS
//!
//! `rust_os` is a kernel running on IA-32 architecture

#![feature(lang_items, asm, const_fn)]
#![no_std]

extern crate rlibc;
extern crate common;

pub mod x86;
pub mod multiboot;
pub mod vga;
pub mod pio;
pub mod paging;
pub mod kheap;
pub mod gdt;
pub mod pic;
pub mod idt;
pub mod timer;
pub mod keyboard;
pub mod ide;
pub mod fs;
pub mod task;
pub mod syscall;

use x86::*;
use multiboot::*;
use vga::*;
use pio::disable_cursor;
use paging::*;
use kheap::*;
use gdt::gdt_init;
use pic::pic_init;
use idt::idt_init;
```

```
36  use timer::*;
37  use fs::*;
38  use task::*;
39  use common::*;
40
41  // exports
42  pub use idt::exception_handler;
43  pub use idt::irq_handler;
44  pub use syscall::syscall_handler;
45
46  #[cfg(test)]
47  mod test;
48
49  /// Entrypoint to the rust code. This function is called by the
    bootstrap code
50  /// contain in bootstrap_asm.s
51  #[no_mangle]
52  pub extern fn kmain(_multiboot_magic: u32, multiboot_info: *mut
    MultibootInfo) {
53      let mboot = unsafe { (*multiboot_info) };
54      vga_init(BG_COLOR, FG_COLOR);
55      println!("Screen initialized.");
56      paging_init();
57      println!("Paging initialized.");
58      kheap_init(mboot.mem_upper);
59      println!("Heap initialized.");
60      gdt_init();
61      println!("GDT initialized.");
62      pic_init();
63      println!("PIC initialized.");
64      idt_init();
65      println!("IDT initialized.");
66      sti();
67      println!("Interrupts unmasked.");
68      timer_init(50);
69      println!("PIT initialized.");
70      set_superblock();
71      println!("Welcome to RustOS!");
72      println!("Available Memory = {} kB", mboot.mem_upper);
73      sleep(3000);
74      exec("splash");
75      exec("shell");
76      print_kmalloc_list();
77      disable_cursor();
78      print!("\nKernel stopped.\nYou can turn off you computer.");
79  }
80
81  #[cfg(not(test))]
```

```
#[lang = "panic_fmt"]
#[no_mangle]
pub extern fn panic_fmt(details: ::core::fmt::Arguments, file: &'static
str, line: u32, column: u32) -> ! {
    println!("panicked at {}, {}:{}:{}", details, file, line, column);
    loop {
        cli();
        halt();
    }
}

#[no_mangle]
pub extern "C" fn __floatundisf() {
    loop {
        cli();
        halt();
    }
}
```

V.XI /kernel/src/keyboard.rs

```
#![allow(dead_code)]

use x86::halt;
use pio::*;
use common::*;

// Keyboard ports
const KEYBOARD_DATA_PORT: u16 = 0x60;
const KEYBOARD_STATE_PORT: u16 = 0x64;

const CIRC_BUFFER_SIZE: usize = 30;

static mut BUFFER: CircBuffer = CircBuffer::new();
static mut SHIFT: bool = false;

struct CircBuffer {
    buffer: [i32;CIRC_BUFFER_SIZE],
    read: usize,
    write: usize,
    count: usize
}

pub fn keyboard_handler() {
    unsafe {
        let state = inb(KEYBOARD_STATE_PORT) & 1;
        if state == 1 {
            let mut key = inb(KEYBOARD_DATA_PORT);
```

```
            if key >> 7 == 0 {
                match key {
                    LEFT_SHIFT => SHIFT = true,
                    RIGHT_SHIFT => SHIFT = true,
                    _ => {
                        if SHIFT {
                            BUFFER.write(SHIFT_KEY_MAP[key as usize] as
i32);
                        } else {
                            BUFFER.write(KEY_MAP[key as usize] as i32);
                        }
                    }
                }
            } else {
                key &= !(1<<7);
                if key == LEFT_SHIFT || key == RIGHT_SHIFT {
                    SHIFT = false;
                }
            }
        }
    }
}

pub fn getc() -> char {
    unsafe {
        let mut data = BUFFER.read();
        while data == -1 {
            halt();
            data = BUFFER.read();
        }
        return data as u8 as char;
    }
}

// Non-blocking call. Return 1 if a key is pressed
pub fn keypressed() -> bool {
    unsafe {
        BUFFER.count > 0
    }
}

impl CircBuffer {
    const fn new() -> CircBuffer {
        CircBuffer {
            buffer: [0;CIRC_BUFFER_SIZE],
            read: 0,
            write: 0,
            count: 0
```

```
        }
    }

    fn write(&mut self, data: i32) {
        self.buffer[self.write] = data;
        self.write = (self.write + 1) % CIRC_BUFFER_SIZE;
        if self.count < CIRC_BUFFER_SIZE {
            self.count += 1;
        }
    }

    fn read(&mut self) -> i32 {
        if self.count > 0 {
            let data = self.buffer[self.read];
            self.read = (self.read + 1) % CIRC_BUFFER_SIZE;
            self.count -= 1;
            return data;
        } else {
            return -1;
        }
    }
}
```

V.XII /kernel/src/kheap.rs

```
#![allow(dead_code)]

use core::mem::size_of;
use rlibc::{memset,memcpy};
use paging::*;
use vga::*;

pub static mut KHEAP_SIZE: usize = 0x1000000;
pub static mut KHEAP_ADDR: u32 = 0;
pub static mut KHEAP_END: u32 = 0;

#[derive(Debug, Clone, Copy)]
#[repr(C, align(16))]
pub struct Header {
    previous: u32,
    next: u32,
    size: usize,
    free: bool
}

macro_rules! align {
    ($size:expr) => {
        if ($size & 0xfffff000) != $size {
```

```
            ($size & 0xfffff000) + 0x1000
        } else {
            $size
        };
    }
}

pub fn kheap_init(ram_size: u32) {
    unsafe {
        KHEAP_ADDR = get_kernel_end();
        KHEAP_SIZE = ((ram_size / 1000 - 1) * 0x100000 -
phys!(KHEAP_ADDR)) as usize;
        KHEAP_END = KHEAP_ADDR + KHEAP_SIZE as u32;
        if (KHEAP_ADDR & 0xfffff000) != KHEAP_ADDR {
            KHEAP_ADDR &= 0xfffff000;
            KHEAP_ADDR += 0x1000;
        }
        let mut entry_addr = 0;
        INITIAL_PD.alloc_frame(&mut entry_addr, &mut phys!(KHEAP_ADDR),
KERNEL_MODE);
        memset(entry_addr as *mut u8, 0, FRAME_SIZE);
        memcpy(entry_addr as *mut u8, Header::null(0,
KHEAP_SIZE).as_ptr(), size_of::<Header>());
    }
}

pub fn kmalloc(size: usize) -> u32 {
    let aligned_size = align!(size) + align!(size_of::<Header>()) -
size_of::<Header>();
    let mut addr = empty_block(aligned_size);
    if kmalloc_table_check(addr, aligned_size) {
        addr = empty_block(aligned_size);
    }
    let mut block = Header::from_ptr(addr as *mut u8);
    if block.size >= aligned_size && block.free {
        if block.next == 0 {
            block.insert_tail(addr, aligned_size);
        } else {
            block.insert(addr, aligned_size);
        }
        addr += size_of::<Header>() as u32;
        unsafe { memset(addr as *mut u8, 0, aligned_size); }
        return addr;
    }
    return 0;
}

pub fn kfree(addr: u32) {
```

```
68      let header_addr = addr - size_of::<Header>() as u32;
69      let mut header = Header::from_ptr(header_addr as *const u8);
70      if !header.free {
71          let mut start_idx = header_addr as usize / FRAME_SIZE + 1;
72          let mut end_idx = (header_addr as usize + header.size) /
    FRAME_SIZE + 1;
73          if header.previous != 0 {
74             let previous = Header::from_ptr(header.previous as *const u8);
75              if previous.free {
76                  start_idx -= 1;
77              }
78          }
79          if header.next != 0 {
80              let next = Header::from_ptr(header.next as *const u8);
81              if next.free {
82                  end_idx += 1;
83              }
84          }
85          header.remove(header_addr);
86          unsafe {
87              // free frames
88              for i in start_idx..end_idx {
89                  INITIAL_PD.free_frame(i);
90              }
91              // free tables
92              for i in (start_idx / TABLE_FSIZE)..(end_idx / TABLE_FSIZE +
    1) {
93                  if INITIAL_PD.table_is_empty(i) {
94                      let mut table_addr = virt!(INITIAL_PD[i] &! 0xfff);
95                      table_addr -= FRAME_SIZE as u32;
96                     let mut table_header = Header::from_ptr(table_addr as
    *const u8);
97                      table_header.remove(table_addr);
98                      INITIAL_PD[i] = 0;
99                  }
100             }
101         }
102     }
103 }
104
105 pub fn umalloc(size: usize) -> u32 {
106     unsafe {
107         let aligned_size = align!(size);
108
109         let pd_backup = if get_cr3() != phys!(INITIAL_PD.tables as u32) {
110             switch_directory(&mut INITIAL_PD);
111             USER_PD
112         } else {
```

```
113             &mut INITIAL_PD as *mut PageDirectory
114         };
115         umalloc_table_check(aligned_size);
116         let mut phys_addr = phys!(kmalloc(size) + (FRAME_SIZE -
    size_of::<Header>()) as u32);
117         switch_directory(pd_backup);
118 
119         let mut virt_addr = 0;
120         let mut tmp = 0;
121         (*USER_PD).alloc_frame(&mut virt_addr, &mut phys_addr,
    USER_MODE);
122         for i in 1..(aligned_size / FRAME_SIZE) {
123             (*USER_PD).alloc_frame(&mut tmp, &mut (phys_addr + (i *
    FRAME_SIZE) as u32), USER_MODE);
124         }
125         return virt_addr;
126     }
127 }
128 
129 pub fn ufree(addr: u32) {
130     unsafe {
131         let frame_idx = addr / FRAME_SIZE as u32;
132         let table_idx = frame_idx as usize / TABLE_FSIZE;
133         let entry_idx = frame_idx as usize % TABLE_FSIZE;
134         let table_ptr = virt!((*USER_PD)[table_idx] &! 0xfff) as *mut
    PageTable;
135         let entry_addr = virt!((*table_ptr)[entry_idx] &! 0xfff);
136         kfree(entry_addr - (FRAME_SIZE - size_of::<Header>()) as u32);
137     }
138 }
139 
140 fn empty_block(size: usize) -> u32 {
141     let mut addr = unsafe { KHEAP_ADDR };
142     let mut block = Header::from_ptr(addr as *mut u8);
143     while block.next != 0 {
144         if block.size >= size && block.free {
145             break;
146         }
147         addr = block.next;
148         block = Header::from_ptr(addr as *mut u8);
149     }
150     return addr;
151 }
152 
153 fn kmalloc_table_check(addr: u32, size: usize) -> bool {
154     unsafe {
155         let total_size = size + size_of::<Header>();
156         let mut alloc = false;
```

```
157         let mut start_idx = addr as usize / TABLE_SIZE;
158         if addr % TABLE_SIZE as u32 != 0 {
159             start_idx += 1;
160         }
161         let mut end_idx = (addr as usize + total_size) / TABLE_SIZE;
162         if Header::from_ptr(addr as *mut u8).next == 0 {
163             end_idx += 1;
164         }
165         let aligned_size = align!(FRAME_SIZE + size_of::<Header>()) -
    size_of::<Header>();
166         for i in start_idx..end_idx {
167             let block_addr = empty_block(aligned_size);
168             let mut block = Header::from_ptr(block_addr as *mut u8);
169             if block.size >= aligned_size && block.free {
170                 let table_addr = INITIAL_PD.new_table(block_addr +
    FRAME_SIZE as u32);
171                 INITIAL_PD[i] = phys!(table_addr) | 0x3;
172                 if block.next == 0 {
173                     block.insert_tail(block_addr, aligned_size);
174                 } else {
175                     block.insert(block_addr, aligned_size);
176                 }
177             }
178             alloc = true;
179         }
180         return alloc;
181     }
182 }
183
184 fn umalloc_table_check(size: usize) {
185     unsafe {
186         let frame_idx = (*USER_PD).mmap_get_free_area(size);
187         let mut start_idx = frame_idx as usize / TABLE_FSIZE;
188         if frame_idx as usize % TABLE_FSIZE == 0 {
189             let table_addr = kmalloc(FRAME_SIZE) + (FRAME_SIZE -
    size_of::<Header>()) as u32;
190             (*USER_PD)[start_idx] = phys!(table_addr) | 0x3 | USER_MODE;
191             start_idx += 1;
192         }
193         if frame_idx as usize + size / FRAME_SIZE >= TABLE_FSIZE {
194             let end_idx = (frame_idx as usize + size / FRAME_SIZE) /
    TABLE_FSIZE;
195             for i in start_idx..end_idx {
196                 let table_addr = kmalloc(FRAME_SIZE) + (FRAME_SIZE -
    size_of::<Header>()) as u32;
197                 (*USER_PD)[i] = phys!(table_addr) | 0x3 | USER_MODE;
198             }
199         }
```

```
200     }
201 }
202 
203 pub fn print_kmalloc_list() {
204     unsafe {
205         let mut addr = KHEAP_ADDR;
206         while addr != 0 {
207             let block = Header::from_ptr(addr as *mut u8);
208             addr = block.next;
209             println!("{:x?}", block);
210         }
211         println!();
212     }
213 }
214 
215 impl Header {
216     fn null(previous: u32, size: usize) -> Header {
217         Header {
218             previous: previous,
219             next: 0,
220             size: size,
221             free: true
222         }
223     }
224 
225     fn insert(&mut self, addr: u32, size: usize) {
226         unsafe {
227             let total_size = size + size_of::<Header>();
228             INITIAL_PD.mmap_set_frame(addr / FRAME_SIZE as u32);
229             if (addr as usize % FRAME_SIZE) + total_size > FRAME_SIZE {
230                 for i in 1..(total_size / FRAME_SIZE + 1) {
231                     let mut phys_addr = phys!(addr + (i * FRAME_SIZE) as
    u32);
232                     let mut tmp = 0;
233                     INITIAL_PD.alloc_frame(&mut tmp, &mut phys_addr,
    KERNEL_MODE);
234                 }
235             }
236             self.free = false;
237             if size == self.size {
238                 memcpy(addr as *mut u8, self.as_ptr(),
    size_of::<Header>());
239             } else {
240                 let tmp = self.next;
241                 let mut tmp_header = Header::from_ptr(tmp as *const u8);
242 
243                 self.size = size;
244                 self.next = addr + total_size as u32;
```

```
245                 let next_block_size = (tmp - self.next) as usize -
size_of::<Header>();
246                 let mut next_header = Header::null(addr, next_block_size);
247                 next_header.next = tmp;
248                 tmp_header.previous = self.next;
249                 memcpy(addr as *mut u8, self.as_ptr(),
size_of::<Header>());
250                 memcpy(self.next as *mut u8, next_header.as_ptr(),
size_of::<Header>());
251                 memcpy(tmp as *mut u8, tmp_header.as_ptr(),
size_of::<Header>());
252             }
253         }
254     }
255
256     fn insert_tail(&mut self, addr: u32, size: usize) {
257         unsafe {
258             let total_size = size + size_of::<Header>();
259             self.size = size;
260             self.free = false;
261             self.next = addr + total_size as u32;
262             INITIAL_PD.mmap_set_frame(addr / FRAME_SIZE as u32);
263             // alloc new frames if need
264             if (addr as usize % FRAME_SIZE) + total_size > FRAME_SIZE {
265                 for i in 1..(total_size / FRAME_SIZE + 1) {
266                     let mut phys_addr = phys!(addr + (i * FRAME_SIZE) as
u32);
267                     let mut tmp = 0;
268                     INITIAL_PD.alloc_frame(&mut tmp, &mut phys_addr,
KERNEL_MODE);
269                 }
270             }
271             let tail_size = (KHEAP_END - self.next) as usize -
size_of::<Header>();
272             let mut tail = Header::null(addr, tail_size);
273             memcpy(addr as *mut u8, self.as_ptr(), size_of::<Header>());
274             memcpy(self.next as *mut u8, tail.as_ptr(),
size_of::<Header>());
275         }
276     }
277
278     fn remove(&mut self, addr: u32) {
279         unsafe {
280             let mut header_addr = addr;
281             if !self.free {
282                 if self.previous != 0 {
283                     let previous = Header::from_ptr(self.previous as
*const u8);
```

```
                    if previous.free {
                        header_addr = self.previous;
                        self.previous = previous.previous;
                        self.size += previous.size + size_of::<Header>();
                    }
                }
                if self.next != 0 {
                    let mut next = Header::from_ptr(self.next as *const u8);
                    if next.free {
                        self.next = next.next;
                        self.size += next.size + size_of::<Header>();
                    }
                    if self.next != 0 {
                        next = Header::from_ptr(self.next as *const u8);
                        if next.previous != header_addr {
                            next.previous = header_addr;
                            memcpy(self.next as *mut u8, next.as_ptr(), size_of::<Header>());
                        }
                    }
                }
                self.free = true;
                memcpy(header_addr as *mut u8, self.as_ptr(), size_of::<Header>());
                INITIAL_PD.mmap_reset_frame(header_addr / FRAME_SIZE as u32);
            }
        }
    }

    fn as_ptr(&mut self) -> *const u8 {
        self as *const Header as *const u8
    }

    fn from_ptr(ptr: *const u8) -> Header {
        unsafe {
            *(ptr as *const Header)
        }
    }
}
```

V.XIII /kernel/src/Makefile

```
ARCH = i386
BUILD_FOLDER = build

LD = gcc
```

```
5  QEMU = qemu-system-$(ARCH)
6  GCC_KERNEL = -static -m32 -ffreestanding -nostdlib
7  LDFLAGS = -T $(LINKER) $(GCC_KERNEL) -Wl,-Map,$(BUILD_FOLDER)/kernel.map
8
9  SRCS = $(wildcard *.s)
10 OBJS = $(patsubst %.s, $(BUILD_FOLDER)/%.o, $(SRCS))
11 RUST = ../target/$(ARCH)-rust_os/debug/libkernel.a
12
13 KERNEL = kernel.elf
14 LINKER = kernel.ld
15
16 GRUB = ../../grub
17 OS = rust_os
18 ISO = $(BUILD_FOLDER)/$(OS).iso
19 IFLAGS = -input-charset utf8 -no-emul-boot -boot-info-table
20
21 FS_FOLDER = ../../tools/MicroFS
22 USER_PATH = ../../user
23 FS = $(BUILD_FOLDER)/fs.img
24 SPLASH = ../../doc/splash.txt
25
26 .PHONY : all run kernel user clean mrproper
27
28 all : $(ISO) $(FS)
29
30 run : $(ISO) $(FS)
31     $(QEMU) -cdrom $(ISO) -hda $(FS)
32
33 kernel : $(BUILD_FOLDER)/$(KERNEL)
34
35 user :
36     $(MAKE) -C $(USER_PATH)
37
38 $(ISO) : | kernel
39     mkdir -p $(BUILD_FOLDER)/isofiles/boot/grub
40     cp $(BUILD_FOLDER)/$(KERNEL) $(BUILD_FOLDER)/isofiles/boot/$(KERNEL)
41     cp -r $(GRUB) $(BUILD_FOLDER)/isofiles/boot
42     genisoimage -R -b boot/grub/stage2_eltorito $(IFLAGS) -o $(@)
   $(BUILD_FOLDER)/isofiles
43     rm -r $(BUILD_FOLDER)/isofiles
44
45 $(BUILD_FOLDER)/$(KERNEL) : $(LINKER) $(OBJS) $(RUST)
46     $(LD) $(LDFLAGS) $(OBJS) $(RUST) -o $@
47
48 $(BUILD_FOLDER)/%.o : %.s
49     mkdir -p $(shell dirname $@)
50     nasm -f elf $< -o $@
51
```

```
$(RUST) :
	$(MAKE) -C ../

$(FS) : $(SPLASH) | user
	cargo run --manifest-path $(FS_FOLDER)/Cargo.toml $@ create MicroFS 1 1000000
	cargo run --manifest-path $(FS_FOLDER)/Cargo.toml $@ add $(SPLASH)
	cargo run --manifest-path $(FS_FOLDER)/Cargo.toml $@ add $(USER_PATH)/build/hello
	cargo run --manifest-path $(FS_FOLDER)/Cargo.toml $@ add $(USER_PATH)/build/demo
	cargo run --manifest-path $(FS_FOLDER)/Cargo.toml $@ add $(USER_PATH)/build/shell
	cargo run --manifest-path $(FS_FOLDER)/Cargo.toml $@ add $(USER_PATH)/build/splash

clean :
	rm -rf $(BUILD_FOLDER)

mrproper : clean
	cargo clean --manifest-path $(FS_FOLDER)/Cargo.toml
	$(MAKE) -C $(USER_PATH) clean
```

V.XIV /kernel/src/multiboot.rs

```rust
#![allow(dead_code)]

#[derive(Debug, Clone, Copy)]
#[repr(C)]
pub struct MultibootAoutSymbolTable {
    pub tabsize: u32,
    pub strsize: u32,
    pub addr: u32,
    pub reserved: u32
}

#[derive(Debug, Clone, Copy)]
#[repr(C)]
pub struct MultibootElfSectionHeaderTable {
    pub num: u32,
    pub size: u32,
    pub addr: u32,
    pub shndx: u32
}

#[derive(Debug, Clone, Copy)]
#[repr(C)]
pub struct MultibootInfo {
```

```
    /* Multiboot info version number */
    pub flags: u32,

    /* Available memory from BIOS (in KB) */
    pub mem_lower: u32,
    pub mem_upper: u32,

    /* "root" partition */
    pub boot_device: u32,

    /* Kernel command line */
    pub cmdline: u32,

    /* Boot-Module list */
    pub mods_count: u32,
    pub mods_addr: u32,

    pub aout_sym: MultibootAoutSymbolTable,
    pub elf_sec: MultibootElfSectionHeaderTable,

    /* Memory Mapping buffer */
    pub mmap_length: u32,
    pub mmap_addr: u32,

    /* Drive Info buffer */
    pub drives_length: u32,
    pub drives_addr: u32,

    /* ROM configuration table */
    pub config_table: u32,

    /* Boot Loader Name */
    pub boot_loader_name: u32,

    /* APM table */
    pub apm_table: u32,

    /* Video */
    pub vbe_control_info: u32,
    pub vbe_mode_info: u32,
    pub vbe_mode: u16,
    pub vbe_interface_seg: u16,
    pub vbe_interface_off: u16,
    pub vbe_interface_len: u16
}

#[derive(Debug, Clone, Copy)]
#[repr(C)]
```

```rust
pub struct MultibootModList {
    /* the memory used goes from bytes 'mod_start' to 'mod_end-1' inclusive */
    pub mod_start: u32,
    pub mod_end: u32,

    /* Module command line */
    pub cmdline: u32,

    /* padding to take it to 16 bytes (must be zero) */
    pub pad: u32
}
```

V.XV /kernel/src/paging_asm.s

```
%include "const.inc"

extern kernel_start
extern kernel_end
extern page_directory
extern kernel_pt

global load_directory
global get_cr3
global get_kernel_start
global get_kernel_end
global get_kernel_page_directory
global get_kernel_page_table

section .text:          ; start of the text (code) section

load_directory:
    push ebp
    mov ebp, esp

    mov eax,[esp+8]     ; Get the pointer to the page directory, passed as a parameter.
    mov cr3, eax

    mov ebx, cr0        ; read current cr0
    or  ebx, 1 << 31    ; set PG
    mov cr0, ebx        ; update cr0

    leave
    ret

get_cr3:
    push ebp
```

```
33      mov ebp, esp
34
35      mov eax, cr3
36
37      leave
38      ret
39
40  get_kernel_start:
41      push ebp
42      mov ebp, esp
43
44      mov eax, kernel_start
45
46      leave
47      ret
48
49  get_kernel_end:
50      push ebp
51      mov ebp, esp
52
53      mov eax, kernel_end
54
55      leave
56      ret
57
58  get_kernel_page_directory:
59      push ebp
60      mov ebp, esp
61
62      mov eax, page_directory
63      add eax, KERNEL_BASE
64
65      leave
66      ret
67
68  get_kernel_page_table:
69      push ebp
70      mov ebp, esp
71
72      mov eax, kernel_pt
73      add eax, KERNEL_BASE
74
75      leave
76      ret
```

V.XVI /kernel/src/paging.rs

```rust
#![allow(dead_code)]
#![macro_use]

use core::mem::size_of;
use core::ops::{Index, IndexMut};
use rlibc::memset;
use vga::*;
use kheap::*;

pub const KERNEL_BASE: u32 = 0xC0000000;
pub const KERNEL_PAGE_NUMBER: u32 = KERNEL_BASE >> 22;

const MMAP_SIZE: usize = 0x20000;
const MEMORY_FSIZE: usize = 0x100000;
pub const TABLE_FSIZE: usize = 0x400;
pub const TABLE_SIZE: usize = 0x400000;
pub const FRAME_SIZE: usize = 0x1000;

pub const KERNEL_MODE: u32 = 0x0;
pub const USER_MODE: u32 = 0x4;

static mut INITIAL_MMAP: [u8;MMAP_SIZE] = [0;MMAP_SIZE];
pub static mut INITIAL_PD: PageDirectory = PageDirectory::null();
pub static mut USER_PD: *mut PageDirectory = 0 as *mut PageDirectory;

#[derive(Clone, Copy)]
#[repr(C, align(4096))]
pub struct PageDirectory {
    pub tables: *mut PageTable,
    pub mmap: *mut [u8;MMAP_SIZE]
}

#[derive(Clone, Copy)]
#[repr(C, align(4096))]
pub struct PageTable {
    pub entries: [u32;TABLE_FSIZE]
}

extern "C" {
    pub fn load_directory(pd_addr: u32);
    pub fn get_cr3() -> u32;
    pub fn get_kernel_start() -> u32;
    pub fn get_kernel_end() -> u32;
    fn get_kernel_page_directory() -> u32;
    fn get_kernel_page_table() -> u32;
}
```

```
47
48  macro_rules! phys {
49      ($addr:expr) => ($addr - KERNEL_BASE);
50  }
51
52  macro_rules! virt {
53      ($addr:expr) => ($addr + KERNEL_BASE);
54  }
55
56  pub fn paging_init() {
57      unsafe {
58         INITIAL_PD.tables = get_kernel_page_directory() as *mut PageTable;
59          INITIAL_PD.mmap = &mut INITIAL_MMAP as *mut [u8;MMAP_SIZE];
60          INITIAL_PD.mmap_set_area(KERNEL_BASE, get_kernel_end());
61      }
62  }
63
64  pub fn switch_directory(pd_ptr: *mut PageDirectory) {
65      unsafe {
66          if pd_ptr as u32 != &INITIAL_PD as *const _ as u32 {
67              USER_PD = pd_ptr;
68              (*USER_PD).update();
69          }
70          load_directory(phys!((*pd_ptr).tables as u32));
71      }
72  }
73
74  impl Index<usize> for PageDirectory {
75      type Output = u32;
76
77      fn index(&self, index: usize) -> &u32 {
78          unsafe { &(*self.tables)[index] }
79      }
80  }
81
82  impl IndexMut<usize> for PageDirectory {
83      fn index_mut(&mut self, index: usize) -> &mut u32 {
84          unsafe { &mut (*self.tables)[index] }
85      }
86  }
87
88  impl PageDirectory {
89      pub const fn null() -> PageDirectory {
90          PageDirectory {
91              tables: 0 as *mut PageTable,
92              mmap: 0 as *mut [u8;MMAP_SIZE]
93          }
94      }
```

```
95
96      pub fn alloc_frame(&mut self, virt: *mut u32, phys: *mut u32, mode:
    u32) -> i32 {
97          unsafe {
98              if *phys < phys!(get_kernel_end()) {
99                  println!("alloc_frame: corrupted address");
100                 return -1;
101             }
102             *virt = if mode == USER_MODE {
103                 self.mmap_alloc_frame(0)
104             } else {
105                 self.mmap_alloc_frame(virt!(*phys))
106             };
107             let frame_idx = *virt / FRAME_SIZE as u32;
108             let table_idx = frame_idx as usize / TABLE_FSIZE;
109             let entry_idx = frame_idx as usize % TABLE_FSIZE;
110
111             let table_addr = virt!(self[table_idx] &! 0xfff);
112             if *virt != table_addr {
113                 let table_ptr = table_addr as *mut PageTable;
114                 (*table_ptr)[entry_idx] = *phys | 0x3 | mode;
115                 memset(*virt as *mut u8, 0, FRAME_SIZE);
116             }
117             return 0;
118         }
119     }
120
121     pub fn free_frame(&mut self, idx: usize) {
122         unsafe {
123             let addr = (idx * FRAME_SIZE) as u32;
124             if addr < get_kernel_end() {
125                 println!("free_frame: corrupted address");
126                 return;
127             }
128             let table_idx = idx / TABLE_FSIZE;
129             let entry_idx = idx % TABLE_FSIZE;
130             self.mmap_reset_frame(idx as u32);
131             let table_ptr = virt!(self[table_idx] &! 0xfff) as *mut
    PageTable;
132             (*table_ptr)[entry_idx] = 0;
133         }
134     }
135
136     pub fn new_directory(&mut self) -> PageDirectory {
137         PageDirectory {
138             tables: (kmalloc(FRAME_SIZE) + (FRAME_SIZE -
    size_of::<Header>()) as u32) as *mut PageTable,
139             mmap: kmalloc(MMAP_SIZE) as *mut [u8;MMAP_SIZE]
```

```
140         }
141     }
142 
143     pub fn free(&mut self) {
144         for i in 0..TABLE_FSIZE {
145             let table_addr = self[i] &! 0xfff;
146             if table_addr != 0 {
147                 kfree(virt!(table_addr) - (FRAME_SIZE -
    size_of::<Header>()) as u32);
148             } else {
149                 break;
150             }
151         }
152         kfree(self.tables as u32 - (FRAME_SIZE - size_of::<Header>()) as
    u32);
153         kfree(self.mmap as u32);
154     }
155 
156     pub fn update(&mut self) {
157         for i in KERNEL_PAGE_NUMBER as usize..TABLE_FSIZE {
158             if self[i] == 0 {
159                 self[i] = unsafe { INITIAL_PD[i] };
160             }
161         }
162     }
163 
164     pub fn new_table(&mut self, addr: u32) -> u32 {
165         unsafe {
166             let phys_addr = phys!(addr);
167             let virt_addr = virt!(phys_addr);
168             let frame_idx = virt_addr / FRAME_SIZE as u32;
169             let table_idx = frame_idx as usize / TABLE_FSIZE;
170             let entry_idx = frame_idx as usize % TABLE_FSIZE;
171             if self[table_idx] == 0 {
172                 // create a temporary table
173                 let mut tmp_table = PageTable::null();
174                 tmp_table[entry_idx] = phys_addr | 0x3;
175                 self[table_idx] = phys!(tmp_table.as_ptr()) | 0x3;
176                 // now that the new table is mapped we can modify it
177                 let table_ptr = virt_addr as *mut PageTable;
178                 memset(virt_addr as *mut u8, 0, size_of::<PageTable>());
179                 (*table_ptr)[entry_idx] = phys_addr | 0x3;
180                 self[table_idx] = phys_addr | 0x3;
181             } else {
182                 let table_ptr = virt!(self[table_idx] &! 0xfff) as *mut
    PageTable;
183                 (*table_ptr)[entry_idx] = phys_addr | 0x3;
184                 memset(virt_addr as *mut u8, 0, size_of::<PageTable>());
```

```
185             }
186             return virt_addr;
187         }
188     }
189 
190     pub fn table_is_empty(&mut self, idx: usize) -> bool {
191         let table_addr = self[idx] &! 0xfff;
192         if table_addr == 0 {
193             return false;
194         }
195         let table_idx = virt!(table_addr) / FRAME_SIZE as u32;
196         self.mmap_reset_frame(table_idx);
197 
198         // let mmap = unsafe { *self.mmap };
199         // let compact_mmap = unsafe { transmute::<[u8;MMAP_SIZE],
    [u32;MMAP_SIZE/4]>(mmap) };
200         let size = TABLE_FSIZE / 8;
201         let mmap_idx = size * idx;
202         let mut is_empty = true;
203 
204         for i in mmap_idx..(mmap_idx+size) {
205             if unsafe { (*self.mmap)[i] } != 0 {
206                 is_empty = false;
207                 break;
208             }
209         }
210         self.mmap_set_frame(table_idx);
211         return is_empty;
212     }
213 
214     pub fn mmap_alloc_frame(&mut self, addr: u32) -> u32 {
215         let frame = if addr == 0 {
216             self.mmap_get_free_frame()
217         } else {
218             addr / FRAME_SIZE as u32
219         };
220         self.mmap_set_frame(frame);
221         return frame * FRAME_SIZE as u32;
222     }
223 
224     pub fn mmap_get_free_frame(&mut self) -> u32 {
225         for i in 0..MEMORY_FSIZE {
226             if self.mmap_frame_state(i as u32) == 0 {
227                 return i as u32;
228             }
229         }
230         return 0;
231     }
```

```
232
233     pub fn mmap_get_free_area(&mut self, size: usize) -> u32 {
234         let mut cnt = 0;
235         let mut start_frame = 0;
236         for i in 0..MEMORY_FSIZE {
237             if self.mmap_frame_state(i as u32) == 0 {
238                 if cnt == 0 {
239                     start_frame = i as u32;
240                 }
241                 cnt += 1;
242             } else {
243                 cnt = 0;
244             }
245             if cnt == size / FRAME_SIZE {
246                 break;
247             }
248         }
249         return start_frame;
250     }
251
252     pub fn mmap_set_frame(&mut self, frame_id: u32) {
253         let mmap_id = frame_id / 8;
254         let bit_offset = frame_id % 8;
255         unsafe { (*self.mmap)[mmap_id as usize] |= 1<<bit_offset; }
256     }
257
258     pub fn mmap_set_area(&mut self, start: u32, end: u32) {
259         let start_frame = start / FRAME_SIZE as u32;
260         let mut end_frame = end / FRAME_SIZE as u32;
261         if end % 0x1000 != 0 {
262             end_frame += 1;
263         }
264         for i in start_frame..end_frame {
265             self.mmap_set_frame(i);
266         }
267     }
268
269     pub fn mmap_reset_frame(&mut self, frame_id: u32) {
270         let mmap_id = frame_id / 8;
271         let bit_offset = frame_id % 8;
272         unsafe { (*self.mmap)[mmap_id as usize] &= !(1<<bit_offset); }
273     }
274
275     pub fn mmap_frame_state(&mut self, frame_id: u32) -> u8 {
276         let mmap_id = frame_id / 8;
277         let bit_offset = frame_id % 8;
278         unsafe { ((*self.mmap)[mmap_id as usize] >> bit_offset) as u8 & 1
}
```

```
    }
}

impl Index<usize> for PageTable {
    type Output = u32;

    fn index(&self, index: usize) -> &u32 {
        &self.entries[index]
    }
}

impl IndexMut<usize> for PageTable {
    fn index_mut(&mut self, index: usize) -> &mut u32 {
        &mut self.entries[index]
    }
}

impl PageTable {
    fn null() -> PageTable {
        PageTable {
            entries: [0;TABLE_FSIZE]
        }
    }

    pub fn from_ptr(addr: u32) -> *mut PageTable {
        addr as *mut PageTable
    }

    pub fn as_ptr(&mut self) -> u32 {
        self as *const PageTable as u32
    }
}
```

V.XVII /kernel/src/pic.rs

```
#![allow(dead_code)]

use pio::*;

// PIC ports
const PIC1_CMD: u16 = 0x20;
const PIC1_DATA: u16 = 0x21;
const PIC2_CMD: u16 = 0xA0;
const PIC2_DATA: u16 = 0xA1;

// End Of Interrupt (reactivate the specified PIC)
const PIC_EOI: u8 = 0x20;

```

```rust
// Initialize the PICs by remapping IRQs 0-15 to 32-47
// More details here: http://wiki.osdev.org/8259_PIC
pub fn pic_init() {
    // By default IRQ 0 to 7 (master PIC) are mapped to interrupts 0-7
    // and IRQ 8 to 15 (slave PIC) are mapped to interrupts 8-15.
    // In protected mode, this scheme conflicts with CPU exceptions wich are
    // reserved by the CPU and mapped to interrupts 0 to 31.
    // Therefore, we remap IRQ 0-7 to interrupts 32-39 and
    // IRQ 8-15 to interrupts 40-47

    unsafe {
        // Restart both PICs
        outb(PIC1_CMD, 0x11);
        outb(PIC2_CMD, 0x11);

        // Remap IRQ [0..7] to [32..39]
        outb(PIC1_DATA, 32);

        // Remap IRQ [8..15] to [40..47]
        outb(PIC2_DATA, 40);

        // Setup cascading
        outb(PIC1_DATA, 0x04);
        outb(PIC2_DATA, 0x02);
        outb(PIC1_DATA, 0x01);
        outb(PIC2_DATA, 0x01);
    }
}

// Send an end-of-interrupt to the PICs.
// An EOI must also be sent to the slave for IRQs > 7
pub fn pic_eoi(irq: u32) {
    unsafe {
        if irq > 7 {
            outb(PIC2_CMD, PIC_EOI);
        }
        outb(PIC1_CMD, PIC_EOI);
    }
}
```

V.XVIII /kernel/src/pio_asm.s

```
global outb
global outw
global inb
global inw

```

```
section .txt

outb:
    push ebp
    mov ebp, esp

    mov word dx, [esp+8]
    mov byte al, [esp+12]
    out dx, al

    mov eax, 0
    leave
    ret

outw:
    push ebp
    mov ebp, esp

    mov word dx, [esp+8]
    mov word ax, [esp+12]
    out dx, ax

    mov eax, 0
    leave
    ret

inb:
    push ebp
    mov ebp, esp

    mov word dx, [esp+8]
    in byte al, dx

    leave
    ret

inw:
    push ebp
    mov ebp, esp

    mov word dx, [esp+8]
    in word ax, dx

    leave
    ret
```

V.XIX /kernel/src/pio.rs

```rust
#![allow(dead_code)]

// CRTC ports
const CRTC_CMD: u16 = 0x3d4;
const CRTC_DATA: u16 = 0x3d5;
// CRTC registers
const CRTC_START_REG: u8 = 0xa;
const CRTC_LOCATION_MSB: u8 = 0xe;
const CRTC_LOCATION_LSB: u8 = 0xf;

extern "C" {
    pub fn outb(port: u16, data: u8);
    pub fn outw(port: u16, data: u16);
    pub fn inb(port: u16) -> u8;
    pub fn inw(port: u16) -> u16;
}

pub fn move_cursor(position: u16) {
    unsafe {
        let pos_msb = ((position >> 8) & 0xff) as u8;
        let pos_lsb = (position & 0xff) as u8;
        outb(CRTC_CMD, CRTC_LOCATION_MSB);
        outb(CRTC_DATA, pos_msb);
        outb(CRTC_CMD, CRTC_LOCATION_LSB);
        outb(CRTC_DATA, pos_lsb);
    }
}

pub fn enable_cursor() {
    unsafe {
        outb(CRTC_CMD, CRTC_START_REG);
        let reg = inb(CRTC_DATA);
        outb(CRTC_CMD, CRTC_START_REG);
        outb(CRTC_DATA, reg & !0x20);
    }
}

pub fn disable_cursor() {
    unsafe {
        outb(CRTC_CMD, CRTC_START_REG);
        let reg = inb(CRTC_DATA);
        outb(CRTC_CMD, CRTC_START_REG);
        outb(CRTC_DATA, reg | 0x20);
    }
}
```

V.XX /kernel/src/syscall_asm.s

```
%include "const.inc"

section .text                       ; start of the text (code) section
align 4                             ; the code must be 4 byte aligned

extern syscall_handler

global _syscall_handler

_syscall_handler:
    ; Save all registers
    push    ebx
    push    ecx
    push    edx
    push    esi
    push    edi
    push    ebp
    push    ds
    push    es
    push    fs
    push    gs

    ; Load kernel data descriptor into all segments
    push    eax
    mov     ax,GDT_KERNEL_DATA_SELECTOR
    mov     ds,ax
    mov     es,ax
    mov     fs,ax
    mov     gs,ax
    pop     eax

    ; Pass the 6 arguments (nb, arg1, etc.) to the syscall_handler
    ; They are in reverse order to match gcc's IA-32 ABI.

    ; use edi (last arg) to store the task's TSS selector the syscall originated
    ; from, so the kernel can properly address the memory associated to the calling task
    ; (by applying the right offset to addresses passed in arguments)
    str     edi  ; store the segment selector from the task register (TR) in edi
    push    edi

    push    esi
    push    edx
    push    ecx
```

```
    push    ebx
    push    eax

    call    syscall_handler

    ; These 6 "pop eax" instructions are only here to balance the pushes
    ; above used to pass the arguments to the syscall_handler function
    pop     ebx
    pop     ebx
    pop     ebx
    pop     ebx
    pop     ebx
    pop     ebx

    ; Restore all registers
    pop     gs
    pop     fs
    pop     es
    pop     ds
    pop     ebp
    pop     edi
    pop     esi
    pop     edx
    pop     ecx
    pop     ebx
    iret
```

V.XXI /kernel/src/syscall.rs

```
#![allow(dead_code)]

use vga::*;
use pio::*;
use timer::*;
use keyboard::*;
use fs::*;
use task::*;
use paging::FRAME_SIZE;
use kheap::*;
use common::*;

extern "C" {
    pub fn _syscall_handler();
}

/// System call handler: call the appropriate system call according to
the nb argument.
/// Called by the assembly code _syscall_handler
```

```rust
#[no_mangle]
pub unsafe extern fn syscall_handler(nb: Syscall, _arg1: u32, _arg2: u32, _arg3: u32, _arg4: u32) -> i32 {
    let addr = 0;
    match nb {
        Syscall::Puts => syscall_puts(addr, _arg1),
        Syscall::Putc => syscall_putc(_arg1),
        Syscall::Exec => syscall_exec(addr, _arg1),
        Syscall::Keypressed => syscall_keypressed(),
        Syscall::Getc => syscall_getc(),
       Syscall::FileStat => syscall_file_stat(addr, _arg1, addr + _arg2),
        Syscall::FileOpen => syscall_file_open(addr, _arg1),
        Syscall::FileClose => syscall_file_close(_arg1),
        Syscall::FileRead => syscall_file_read(_arg1, addr + _arg2, _arg3),
        Syscall::FileSeek => syscall_file_seek(_arg1, _arg2),
        Syscall::FileIterator => syscall_file_iterator(addr + _arg1),
       Syscall::FileNext => syscall_file_next(addr, _arg1, addr + _arg2),
        Syscall::GetTicks => syscall_get_ticks(),
        Syscall::Sleep => syscall_sleep(_arg1),
        Syscall::SetCursor => syscall_set_cursor(_arg1, _arg2),
        Syscall::GetCursor => syscall_get_cursor(addr + _arg1, addr + _arg2),
        Syscall::CursorDisable => syscall_cursor_disable(_arg1),
        Syscall::CopyScr => syscall_copy_scr(addr + _arg1),
        Syscall::AllocFrame => syscall_alloc_frame(),
        Syscall::FreeFrame => syscall_free_frame(_arg1),
    }
}

unsafe fn syscall_puts(base_addr: u32, string_offset: u32) -> i32 {
    let mut string = *((base_addr + string_offset) as *mut String);
    string.offset(base_addr);
    vga_write_str(string.to_string());
    return 0;
}

unsafe fn syscall_putc(c: u32) -> i32 {
    vga_write_byte(c as u8);
    return 0;
}

unsafe fn syscall_exec(base_addr: u32, string_offset: u32) -> i32 {
    let mut string = *((base_addr + string_offset) as *mut String);
    string.offset(base_addr);
    exec(string.to_string()) as i32
}

```

```
64  unsafe fn syscall_keypressed() -> i32 {
65      keypressed() as i32
66  }
67
68  unsafe fn syscall_getc() -> i32 {
69      getc() as i32
70  }
71
72  unsafe fn syscall_file_stat(base_addr: u32, string_offset: u32,
    stat_addr: u32) -> i32 {
73      let mut string = *((base_addr + string_offset) as *mut String);
74      string.offset(base_addr);
75      let stat = stat_addr as *mut Stat;
76      *stat = Stat::new(string.to_string());
77      if (*stat).start == 0 {
78          return -1;
79      }
80      return 0;
81  }
82
83  unsafe fn syscall_file_open(base_addr: u32, string_offset: u32) -> i32 {
84      let mut string = *((base_addr + string_offset) as *mut String);
85      string.offset(base_addr);
86      file_open(string.to_string())
87  }
88
89  unsafe fn syscall_file_close(fd: u32) -> i32 {
90      file_close(fd as i32)
91  }
92
93  unsafe fn syscall_file_read(fd: u32, buf_addr: u32, n: u32) -> i32 {
94      file_read(fd as i32, buf_addr as *mut u8, n as usize)
95  }
96
97  unsafe fn syscall_file_seek(fd: u32, offset: u32) -> i32 {
98      file_seek(fd as i32, offset as usize)
99  }
100
101 unsafe fn syscall_file_iterator(it_addr: u32) -> i32 {
102     let it = it_addr as *mut FileIterator;
103     *it = FileIterator::new();
104     return 0;
105 }
106
107 unsafe fn syscall_file_next(base_addr: u32, string_offset: u32, it_addr:
    u32) -> i32 {
108     let bytes = (base_addr + string_offset) as *mut u8;
109     let it = it_addr as *mut FileIterator;
```

```
110     (*it).next(bytes) as i32
111 }
112
113 unsafe fn syscall_get_ticks() -> i32 {
114     get_ticks() as i32
115 }
116
117 unsafe fn syscall_sleep(ms: u32) -> i32 {
118     sleep(ms);
119     return 0;
120 }
121
122 unsafe fn syscall_set_cursor(x: u32, y: u32) -> i32 {
123     vga_set_cursor(x as usize, y as usize);
124     return 0;
125 }
126
127 unsafe fn syscall_get_cursor(x_addr: u32, y_addr: u32) -> i32 {
128     let cursor = vga_get_cursor();
129     *(x_addr as *mut u32) = cursor.0 as u32;
130     *(y_addr as *mut u32) = cursor.1 as u32;
131     return 0;
132 }
133
134 unsafe fn syscall_cursor_disable(cd: u32) -> i32 {
135     if cd == 0 {
136         enable_cursor();
137     } else {
138         disable_cursor();
139     }
140     return 0;
141 }
142
143 unsafe fn syscall_copy_scr(scr_addr: u32) -> i32 {
144     vga_copy_scr(scr_addr as *const FrameBuffer);
145     return 0;
146 }
147
148 unsafe fn syscall_alloc_frame() -> i32 {
149     umalloc(FRAME_SIZE) as i32
150 }
151
152 unsafe fn syscall_free_frame(addr: u32) -> i32 {
153     ufree(addr);
154     return 0;
155 }
```

V.XXII /kernel/src/task_asm.s

```
global task_ltr
global task_switch

section .data
tss_sel_offs dd 0  ; must always be 0
tss_sel_seg  dw 0  ; overwritten by the task_switch function

section .text:                    ; start of the text (code) section
align 4                           ; the code must be 4 byte aligned

; Load the task register with the TSS selector passed in argument.
; The TSS selector represents the offset of the TSS descriptor in the GDT table.
; void load_task_register(uint16_t tss_selector);
; NOTE: The GDT must be loaded before issuing the ltr instruction!
;
; void task_ltr(uint16_t tss_selector);
task_ltr:
    mov     eax,[esp+4]
    ltr     ax
    ret

; Call the task specified by the tss selector in argument.
; When the CPU switches to the new task, it automatically loads the task register
; with the new task (ltr instruction) and the LDT from the tss.ldt_selector field.
;
; void task_switch(uint16_t tss_selector)
task_switch:
    mov     ax,[esp+4]  ; get the TSS selector passed in argument (16 bits)
    ; rewrite the segment to jump to with the tss selector passed in argument
    mov        ecx,tss_sel_seg
    mov     [ecx],ax
    call    far [ecx-4]
    ret
```

V.XXIII /kernel/src/task.rs

```
#![allow(dead_code)]

use core::mem::size_of;
use x86::*;
use gdt::*;
```

```
6  use paging::*;
7  use fs::*;
8  use vga::*;
9  use kheap::*;
10 use common::*;
11
12 pub const TASKS_NB: usize = 8;
13 pub const STACK_SIZE: usize = 0x10000;
14
15 pub static mut INITIAL_TSS: Tss = Tss::new();
16 pub static mut INITIAL_TSS_KERNEL_STACK: [u8;STACK_SIZE] =
   [0;STACK_SIZE];
17 pub static mut TASKS: [Task;TASKS_NB] = [Task::new();TASKS_NB];
18
19 #[derive(Clone, Copy)]
20 #[repr(C)]
21 pub struct Task {
22     pub tss: Tss,
23     pub tss_selector: u16,
24     pub kernel_stack: [u8;STACK_SIZE],
25     pub is_free: bool,
26     pub pd: PageDirectory
27 }
28
29 #[derive(Debug, Clone, Copy)]
30 #[repr(C, packed)]
31 pub struct Tss {
32     previous_task_link: u16, reserved0: u16,
33     esp0: u32,
34     ss0: u16, reserved1: u16,
35     esp1: u32,
36     ss1: u16, reserved2: u16,
37     esp2: u32,
38     ss2: u16, reserved3: u16,
39     cr3: u32,
40     eip: u32, eflags: u32, eax: u32, ecx: u32, edx: u32,
41     ebx: u32, esp: u32, ebp: u32, esi: u32, edi: u32,
42     es: u16, reserved4: u16,
43     cs: u16, reserved5: u16,
44     ss: u16, reserved6: u16,
45     ds: u16, reserved7: u16,
46     fs: u16, reserved8: u16,
47     gs: u16, reserved9: u16,
48     ldt_selector: u16, reserved10: u16,
49     reserved11: u16,
50     iomap_base_addr: u16  // adresse (relative to byte 0 of the TSS) of
   the IO permission bitmap
51 }
```

```
52
53  extern "C" {
54      fn task_ltr(tss_selector: u16);
55      fn task_switch(tss_selector: u16);
56  }
57
58  pub fn tasks_init() {
59      unsafe {
60          INITIAL_TSS.ss0 = GDT_KERNEL_DATA_SELECTOR as u16;
61          INITIAL_TSS.esp0 = &INITIAL_TSS_KERNEL_STACK as *const _ as u32 +
    STACK_SIZE as u32;
62          INITIAL_TSS.cr3 = phys!(INITIAL_PD.tables as u32);
63          GDT[5] = GdtEntry::make_tss(&INITIAL_TSS as *const _ as u32,
    DPL_KERNEL);
64          task_ltr(GDT[5].to_selector() as u16);
65
66          for task in &mut TASKS {
67              task.setup();
68          }
69      }
70  }
71
72  pub fn exec(filename: &str) -> i8 {
73      let idx = free_task();
74      if idx != -1 {
75          unsafe {
76              let fd = file_open(filename);
77              if fd != -1 {
78                  let stat = Stat::new(filename);
79                  if file_type(fd) != TYPE_EXEC {
80                      println!("exec: {}: not an executable", filename);
81                      return -1;
82                  }
83                  // Create new directory using initial directory
84                  let pd_backup = if get_cr3() != phys!(INITIAL_PD.tables
    as u32) {
85                      switch_directory(&mut INITIAL_PD);
86                      USER_PD
87                  } else {
88                      &mut INITIAL_PD as *mut PageDirectory
89                  };
90                  TASKS[idx as usize].pd = INITIAL_PD.new_directory();
91                  switch_directory(&mut TASKS[idx as usize].pd);
92
93                  // Alloc frames starting at address 0
94                  // Additional frames are allocated for the stack
95                  let code_addr = umalloc(stat.size);
96                  let stack_addr = umalloc(STACK_SIZE);
```

```
97                  file_read(fd, code_addr as *mut u8, stat.size);
98
99                  // Setup task with page directory previously allocated
100                 TASKS[idx as usize].is_free = false;
101                 TASKS[idx as usize].tss.eip = 0;
102                 TASKS[idx as usize].tss.esp = stack_addr + STACK_SIZE as
    u32;
103                 TASKS[idx as usize].tss.ebp = stack_addr + STACK_SIZE as
    u32;
104                 TASKS[idx as usize].tss.cr3 = phys!(TASKS[idx as
    usize].pd.tables as u32);
105
106                 task_switch(TASKS[idx as usize].tss_selector as u16);
107                 switch_directory(&mut INITIAL_PD);
108                 // re-load original directory and free memory
109                 TASKS[idx as usize].is_free = true;
110                 ufree(code_addr);
111                 ufree(stack_addr);
112                 switch_directory(pd_backup);
113                 TASKS[idx as usize].pd.free();
114                 return 0;
115             } else {
116                 println!("exec: {}: not found", filename);
117             }
118         }
119     } else {
120         println!("exec: no free task slot found");
121     }
122     return -1;
123 }
124
125 fn free_task() -> i8 {
126     unsafe {
127         let mut cnt = 0;
128         for task in TASKS.iter() {
129             if task.is_free {
130                 return cnt;
131             }
132             cnt += 1;
133         }
134         return -1;
135     }
136 }
137
138 impl Task {
139     const fn new() -> Task {
140         Task {
141             tss: Tss::new(),
```

```
142             tss_selector: 0,
143             kernel_stack: [0;STACK_SIZE],
144             is_free: true,
145             pd: PageDirectory::null()
146         }
147     }
148 
149     unsafe fn setup(&mut self) {
150         let idx = ((self as *mut _ as usize) - (&TASKS as *const _ as
    usize)) / size_of::<Task>();
151         // Add the task's TSS to the GDT
152         let tss = &self.tss as *const _ as u32;
153         GDT[GDT_SIZE + idx] = GdtEntry::make_tss(tss, DPL_KERNEL);
154 
155         // Initialize the TSS fields
156         self.tss_selector = GDT[GDT_SIZE + idx].to_selector() as u16;
157 
158         // Code and data segment selectors are in the LDT
159         let cs = (GDT_USER_CODE_SELECTOR | DPL_USER) as u32;
160         let ds = (GDT_USER_DATA_SELECTOR | DPL_USER) as u32;
161         self.tss.cs = cs as u16;
162         self.tss.ds = ds as u16;
163         self.tss.es = self.tss.ds;
164         self.tss.fs = self.tss.ds;
165         self.tss.gs = self.tss.ds;
166         self.tss.ss = self.tss.ds;
167         self.tss.eflags = 512;  // Activate hardware interrupts (bit 9)
168 
169         // Task's kernel stack
170         self.tss.ss0 = GDT_KERNEL_DATA_SELECTOR as u16;
171         self.tss.esp0 = (&self.kernel_stack as *const _ as usize +
    STACK_SIZE) as u32;
172     }
173 }
174 
175 impl Tss {
176     const fn new() -> Tss {
177         Tss {
178             previous_task_link: 0, reserved0: 0,
179             esp0: 0,
180             ss0: 0, reserved1: 0,
181             esp1: 0,
182             ss1: 0, reserved2: 0,
183             esp2: 0,
184             ss2: 0, reserved3: 0,
185             cr3: 0,
186             eip: 0, eflags: 0, eax: 0, ecx: 0, edx: 0,
187             ebx: 0, esp: 0, ebp: 0, esi: 0, edi: 0,
```

```
            es: 0, reserved4: 0,
            cs: 0, reserved5: 0,
            ss: 0, reserved6: 0,
            ds: 0, reserved7: 0,
            fs: 0, reserved8: 0,
            gs: 0, reserved9: 0,
            ldt_selector: 0, reserved10: 0,
            reserved11: 0,
            iomap_base_addr: 0
        }
    }
}
```

V.XXIV /kernel/src/timer.rs

```
#![allow(dead_code)]

use x86::halt;
use core::u32;
use pio::outb;

// PIT ports
const PIT_CMD: u16 = 0x43;
const PIT_CANAL_0: u16 = 0x40;
const PIT_DIV_REPEAT: u8 = 0x36;

pub const MIN_FREQ: u32 = 19;
pub const MAX_FREQ: u32 = 1193180;

static mut TIMER: Timer = Timer { freq: 0, ticks: 0 };

pub fn timer_init(freq_hz: u32) {
    unsafe { TIMER.init(freq_hz) }
}

pub fn timer_handler() {
    unsafe { TIMER.ticks+=1 }
}

pub fn get_freq() -> u32 {
    unsafe { return TIMER.freq }
}

pub fn get_ticks() -> u32{
    unsafe { return TIMER.ticks }
}

pub fn sleep(ms: u32) {
```

```
    let duration = get_ticks() + (ms * unsafe { TIMER.freq } / 1000);
    loop {
        if get_ticks() >= duration {
            break;
        }
        halt();
    }
}

struct Timer {
    freq: u32,
    ticks: u32
}
impl Timer {
    pub fn init(&mut self, freq_hz: u32) {
        match freq_hz {
            0...MIN_FREQ => self.freq = MIN_FREQ,
            MAX_FREQ...u32::MAX => self.freq =  MAX_FREQ,
            _ => self.freq = freq_hz
        }

        #[cfg(not(test))]
        unsafe {
            let div = MAX_FREQ / self.freq;
            // divisor selection and repetition mode
            outb(PIT_CMD, PIT_DIV_REPEAT);
            // divisor LSB on canal 0
            outb(PIT_CANAL_0, (div & 0xFF) as u8);
            // divisor MSB on canal 0
            outb(PIT_CANAL_0, (div >> 8) as u8);
        }
    }

    pub fn get_freq(&mut self) -> u32 {
        return self.freq;
    }

    pub fn get_ticks(&mut self) -> u32{
        return self.ticks;
    }
}
```

V.XXV /kernel/src/vga.rs

```
#![allow(dead_code)]
#![macro_use]

use core::fmt::{Error, Write, Arguments};
```

```
5  use pio::*;
6  use common::*;
7
8  const TAB_SIZE: usize = 4;
9
10 static mut SCREEN: Screen = Screen {
11     buffer: 0xC00B8000 as *mut _,
12     attribute: ColorAttribute::new(BG_COLOR, FG_COLOR),
13     cursor_x: 0,
14     cursor_y: 0
15 };
16
17 struct Screen {
18     buffer: *mut FrameBuffer,
19     attribute: ColorAttribute,
20     cursor_x: usize,
21     cursor_y: usize
22 }
23
24 macro_rules! print {
25     ($($arg:tt)*) => (vga_write_fmt(format_args!($($arg)*)));
26 }
27
28 macro_rules! println {
29     () => (print!("\n"));
30     ($fmt:expr) => (print!(concat!($fmt, "\n")));
31     ($fmt:expr, $($arg:tt)*) => (print!(concat!($fmt, "\n"), $($arg)*));
32 }
33
34 pub fn vga_init(background: Color, foreground: Color) {
35     unsafe {
36         SCREEN.set_color(background, foreground);
37         SCREEN.clear();
38     }
39 }
40
41 pub fn vga_write_byte(byte: u8) {
42     unsafe {
43         SCREEN.write_byte(byte);
44     }
45 }
46
47 pub fn vga_write_str(s: &str) {
48     unsafe {
49         SCREEN.write_str(s);
50     }
51 }
52
```

```
53  pub fn vga_write_fmt(args: Arguments) {
54      unsafe {
55          SCREEN.write_fmt(args).ok();
56      }
57  }
58
59  pub fn vga_clear() {
60      unsafe { SCREEN.clear(); }
61  }
62
63  pub fn vga_set_cursor(x: usize, y: usize) {
64      unsafe { SCREEN.set_cursor(x, y); }
65  }
66
67  pub fn vga_get_cursor() -> (usize, usize) {
68      unsafe {
69          return (SCREEN.cursor_x, SCREEN.cursor_y);
70      }
71  }
72
73  pub fn vga_set_color(background: Color, foreground: Color) {
74      unsafe { SCREEN.set_color(background, foreground); }
75  }
76
77  pub fn vga_copy_scr(scr: *const FrameBuffer) {
78      unsafe {
79          for i in 0..BUFFER_HEIGHT {
80              for j in 0..BUFFER_WIDTH {
81                  (*SCREEN.buffer)[i][j] = (*scr)[i][j];
82              }
83          }
84      }
85  }
86
87  impl Write for Screen {
88      fn write_str(&mut self, s: &str) -> Result<(), Error> {
89          self.write_str(s);
90          Ok(())
91      }
92  }
93
94  impl Screen {
95      fn clear(&mut self) {
96          unsafe {
97              for i in 0..BUFFER_HEIGHT {
98                  for j in 0..BUFFER_WIDTH {
99                      (*self.buffer)[i][j] = Character::new(0,
    self.attribute);
```

```
                }
            }
            self.set_cursor(0, 0);
        }
    }

    fn write_byte(&mut self, byte: u8) {
        if byte == b'\n' || self.cursor_x >= BUFFER_WIDTH {
            if self.cursor_y == BUFFER_HEIGHT-1 {
                self.shift_up();
                self.cursor_x = 0;
            } else {
                self.cursor_x = 0;
                self.cursor_y += 1;
            }
        }
        match byte {
            b'\0' => return,
            b'\n' => (),
            b'\t' => {
                for _i in 0..TAB_SIZE {
                    self.write_byte(b' ');
                }
            }
            0x8 => {
                if self.cursor_x > 0 {
                    self.cursor_x -= 1;
                } else if self.cursor_y > 0 {
                    self.cursor_y -= 1;
                    self.cursor_x = BUFFER_WIDTH-1;
                }
                unsafe { (*self.buffer)[self.cursor_y][self.cursor_x] =
Character::new(b'\0', self.attribute); }
            }
            _ => {
                unsafe { (*self.buffer)[self.cursor_y][self.cursor_x] =
Character::new(byte, self.attribute); }
                self.cursor_x += 1;
            }
        }
        move_cursor(self.get_pos());
    }

    fn write_str(&mut self, buf: &str) {
        for byte in buf.bytes() {
            self.write_byte(byte);
        }
    }
```

```
146
147     fn shift_up(&mut self) {
148         unsafe {
149             for i in 0..BUFFER_HEIGHT-1 {
150                 for j in 0..BUFFER_WIDTH {
151                     (*self.buffer)[i][j] = (*self.buffer)[i+1][j];
152                 }
153             }
154             for j in 0..BUFFER_WIDTH {
155                 (*self.buffer)[BUFFER_HEIGHT-1][j] = Character::new(0,
    self.attribute);
156             }
157         }
158     }
159
160     fn get_pos(&mut self) -> u16 {
161         (self.cursor_y*BUFFER_WIDTH+self.cursor_x) as u16
162     }
163
164     fn get_color(&mut self) -> ColorAttribute {
165         return self.attribute;
166     }
167
168     fn set_color(&mut self, background: Color, foreground: Color) {
169         self.attribute = ColorAttribute::new(background, foreground);
170     }
171
172     fn set_cursor(&mut self, x: usize, y: usize) {
173         self.cursor_x = x;
174         self.cursor_y = y;
175         move_cursor(self.get_pos());
176     }
177 }
```

V.XXVI /kernel/src/x86.rs

```
1  #![allow(dead_code)]
2
3  // Privilege levels
4  pub const  DPL_USER: u8 = 0x3;
5  pub const  DPL_KERNEL: u8 = 0x0;
6
7  // Selectors
8  pub const  LDT_SELECTOR: u8 = 0x4;
9
10 // Descriptor types for code and data segments
11 pub const  TYPE_DATA_READONLY: u8 = 1;
12 pub const  TYPE_DATA_READWRITE: u8 = 3;
```

```rust
pub const  TYPE_CODE_EXECONLY: u8 = 9;
pub const  TYPE_CODE_EXECREAD: u8 = 11;

// Descriptor types for system segments and gates
pub const  TYPE_LDT: u8 = 2;
pub const  TYPE_TASK_GATE: u8 = 5;
pub const  TYPE_TSS: u8 = 9;
pub const  TYPE_CALL_GATE: u8 = 12;
pub const  TYPE_TRAP_GATE: u8 = 15;
pub const  TYPE_INTERRUPT_GATE: u8 = 14;

// Descriptor system bit (S)
// For code or data segments
pub const  S_CODE_OR_DATA: u8 = 1;
// For TSS segment, LDT, call gate, interrupt gate, trap gate, task gate
pub const  S_SYSTEM: u8 = 0;

// D/B bit
pub const  DB_SEG: u8 = 1;
pub const  DB_SYS: u8 = 0;

// kernel code and data selectors in the GDT
pub const  GDT_KERNEL_CODE_SELECTOR: u8 = 0x08;
pub const  GDT_KERNEL_DATA_SELECTOR: u8 = 0x10;
pub const  GDT_USER_CODE_SELECTOR: u8 = 0x18;
pub const  GDT_USER_DATA_SELECTOR: u8 = 0x20;

// Disable hardware interrupts.
pub fn cli() {
    unsafe { asm!("cli"); }
}

// Enable hardware interrupts.
pub fn sti() {
    unsafe { asm!("sti"); }
}

// Halt the processor.
// External interrupts wake up the CPU, hence the cli instruction.
pub fn halt() {
    unsafe { asm!("hlt"); }
}
```

V.XXVII /kernel/Cargo.toml

```toml
[package]
name = "rust_os"
version = "0.1.0"
```

```
authors = ["orpheeantoniadis <orphee.antoniadis@gmail.com>"]

[lib]
name = "kernel"
path = "src/kernel.rs"
crate-type = ["staticlib"]

[dependencies]
rlibc = "1.0"
common = { path = "../common" }

[profile.release]
lto = true
panic = 'abort'
```

V.XXVIII /kernel/i386-rust_os.json

```
{
  "llvm-target": "i386-unknown-none",
  "data-layout": "e-m:e-p:32:32-f64:32:64-f80:32-n8:16:32-S128",
  "linker-flavor": "gcc",
  "target-endian": "little",
  "target-pointer-width": "32",
  "target-c-int-width": "32",
  "arch": "x86",
  "os": "none",
  "disable-redzone": true,
  "features": "-mmx,-sse,+soft-float",
  "panic-strategy": "abort"
}
```

V.XXIX /kernel/Makefile

```
ARCH = i386
CC = xargo
TARGET = $(ARCH)-rust_os
FLAGS = -v --target $(TARGET)
SOURCE_PATH = src/

.PHONY: all build run test clean mrproper

all: build

build:
    RUST_TARGET_PATH=$(shell pwd) $(CC) build $(FLAGS)
    $(MAKE) -C $(SOURCE_PATH)

```

```
15 run:
16     $(MAKE) -C $(SOURCE_PATH) run
17
18 test:
19     cargo test
20
21 doc :
22     $(CC) doc --open
23
24 clean:
25     $(MAKE) -C $(SOURCE_PATH) clean
26     $(CC) clean
27
28 mrproper:
29     $(MAKE) -C $(SOURCE_PATH) mrproper
30     $(CC) clean
```

VI /tools

VI.I /tools/MircroFS/src/micro_fs/add.rs

```
/tools/MircroFS/src/micro_fs/add.rs
```

VI.II /tools/MircroFS/src/micro_fs/create.rs

```
/tools/MircroFS/src/micro_fs/create.rs
```

VI.III /tools/MircroFS/src/micro_fs/del.rs

```
/tools/MircroFS/src/micro_fs/del.rs
```

VI.IV /tools/MircroFS/src/micro_fs/info.rs

```
/tools/MircroFS/src/micro_fs/info.rs
```

VI.V /tools/MircroFS/src/micro_fs/list.rs

```
/tools/MircroFS/src/micro_fs/list.rs
```

VI.VI /tools/MircroFS/src/micro_fs/mod.rs

```
/tools/MircroFS/src/micro_fs/mod.rs
```

VI.VII /tools/MircroFS/src/micro_fs/save.rs

```
/tools/MircroFS/src/micro_fs/save.rs
```

VI.VIII /tools/MircroFS/src/micro_fs/utils.rs

```
/tools/MircroFS/src/micro_fs/utils.rs
```

VI.IX /tools/MircroFS/src/cli.yml

```
/tools/MircroFS/src/cli.yml
```

VI.X /tools/MircroFS/src/lib.rs

```
/tools/MircroFS/src/lib.rs
```

VI.XI /tools/MircroFS/src/main.rs

```
1 /tools/MircroFS/src/main.rs
```

VI.XII /tools/MircroFS/tests/block_size_1.rs

```
1 /tools/MircroFS/tests/block_size_1.rs
```

VI.XIII /tools/MircroFS/Cargo.toml

```
1 /tools/MircroFS/Cargo.toml
```

VII /user

VII.I /user/demo/src/demo.rs

```
#![no_std]

extern crate ulibc;
use ulibc::*;
use io::*;
use mem::*;

#[no_mangle]
pub extern fn main() {
    clear();
    puts("Starting demo.\n");

    println!("\nIO demo :\n");
    println!("Executing hello app..");
    exec("hello");
    println!("Waiting on keypressed..");
    while keypressed() == 0 {}
    getc();
    println!("Waiting 1 sec..");
    sleep(1000);
    println!("Waiting on getc..");
    let key = getc();
    println!("{} pressed.", key as u8 as char);
    sleep(3000);
    clear();

    println!("File system demo :\n");
    println!("Opening file splash.txt..");
    let fd = file_open("splash.txt");
    if fd != -1 {
        println!("Reading file splash.txt..");
        let mut data = [0;MAX_STR_LEN];
        file_read(fd as u32, &mut data[0], MAX_STR_LEN as u32);
        println!("{}", bytes_to_str(&data));
        println!("Closing file splash.txt..");
        file_close(fd as u32);
    }
    println!("Iterating all the files..", );
    let it = file_iterator();
    let mut bytes = [0;MAX_FILENAME_LENGTH];
    while file_next(&bytes[0], &it) != -1 {
        {
            let filename = bytes_to_str(&bytes);
            println!("{} {}", filename, file_stat(filename).size);
        }
```

```
        bytes = [0;MAX_FILENAME_LENGTH];
    }
    sleep(3000);
    clear();

    println!("Memory management demo :\n");
    unsafe {
        println!("Allocating 1M on the heap..", );
        let addr1 = malloc(0x100000);
        *(addr1 as *mut u8) = 42;
        println!("addr1 = 0x{:x}, [addr1] = 0x{:x}", addr1, *(addr1 as
*mut u8));
        println!("Allocating 256 bytes on the heap..");
        let addr2 = malloc(0x100);
        *(addr2 as *mut u8) = 0x42;
        println!("addr2 = 0x{:x}, [addr2] = 0x{:x}", addr2, *(addr2 as
*mut u8));
        println!("Freeing addr1..");
        free(addr1);
        println!("Allocating 4KB on the heap..");
        let addr3 = malloc(0x1000);
        *(addr3 as *mut u8) = 12;
        println!("addr3 = 0x{:x}, [addr3] = 0x{:x}", addr3, *(addr3 as
*mut u8));
        println!("Alloc list state :");
        print_kmalloc_list();
        println!("Freeing addr2..");
        free(addr2);
        println!("Freeing addr3..");
        free(addr3);
        println!("Alloc list state :");
        print_kmalloc_list();
    }
    println!("Ok.");
}
```

VII.II /user/demo/src/hello.rs

```
/user/demo/src/hello.rs
```

VII.III /user/demo/src/shell.rs

```
/user/demo/src/shell.rs
```

VII.IV /user/skeleton/src/skeleton.rs

```rust
#![no_std]

extern crate ulibc;
use ulibc::*;
use io::*;

#[no_mangle]
pub extern fn main() {
    println!("Hello world!");
}
```

VII.V /user/skeleton/Cargo.toml

```
[package]
name = "skeleton"
version = "0.1.0"
authors = ["orpheeantoniadis <orphee.antoniadis@gmail.com>"]

[lib]
name = "skeleton"
path = "src/skeleton.rs"
crate-type = ["staticlib"]

[dependencies]
ulibc = { path = "../ulibc" }

[profile.release]
lto = true
panic = 'abort'
```

VII.VI /user/skeleton/Makefile

```
ARCH = i386
TARGET = $(ARCH)-app

CC = xargo
RFLAGS = "-C relocation-model=static -C opt-level=3"
XFLAGS = --release -vv --target $(TARGET)

.PHONY : all build test clean

all : build

build :
    RUST_TARGET_PATH=$(shell pwd)/.. RUSTFLAGS=$(RFLAGS) $(CC) build
$(XFLAGS)
```

```
clean :
    $(CC) clean
```

VII.VII /user/splash/src/splash.rs

```
#![no_std]

extern crate ulibc;
use ulibc::*;
use io::*;

#[no_mangle]
pub extern fn main() {
    cursor_disable(true);
    clear();
    set_cursor(22,10);
    let fd = file_open("splash.txt") as u32;
    let mut data = [0;MAX_STR_LEN];
    file_read(fd, &mut data[0], MAX_STR_LEN as u32);
    for byte in bytes_to_str(&data).bytes() {
        putc(byte);
        if byte == b'\n' {
            let cursor = (0,0);
            get_cursor(&cursor.0, &cursor.1);
            set_cursor(22,cursor.1);
        }
    }
    file_close(fd);
    sleep(5000);
    clear();
    cursor_disable(false);
}
```

VII.VIII /user/ulibc/src/curses.rs

```
use io::*;

static mut SCR: FrameBuffer =
[[Character::null();BUFFER_WIDTH];BUFFER_HEIGHT];

pub struct Image {
    content: *const u8,
    width: usize,
    height: usize
}

```

```rust
pub struct Window {
    img: Image,
    x: usize,
    y: usize,
    color: ColorAttribute
}

pub fn init_scr() {
    unsafe {
        copy_scr(&SCR);
        set_cursor(0, 0);
        cursor_disable(true);
    }
}

pub fn destroy_scr() {
    unsafe {
        SCR = [[Character::null();BUFFER_WIDTH];BUFFER_HEIGHT];
        copy_scr(&SCR);
        set_cursor(0, 0);
        cursor_disable(false);
    }
}

pub fn update_scr() {
    unsafe {
        copy_scr(&SCR);
    }
}

pub fn display_window(w: Window) {
    unsafe {
        for i in 0..w.img.height {
            for j in 0..w.img.width {
                let ascii = *w.img.content.offset((j + i * w.img.width) as isize);
                SCR[i+w.x][j+w.y] = Character::new(ascii, w.color);
            }
        }
    }
    update_scr();
}

impl Image {
    pub fn new(data: &str, width: usize, height: usize) -> Image {
        Image {
            content: data.as_ptr(),
            width: width,
```

```
            height: height
        }
    }
}

impl Window {
    pub fn new(img: Image, x: usize, y: usize, background: Color,
foreground: Color) -> Window {
        Window {
            img: img,
            x: x,
            y: y,
            color: ColorAttribute::new(background, foreground)
        }
    }
}
```

VII.IX /user/ulibc/src/io.rs

```
#![macro_use]

use core::fmt::{Error, Write, Arguments};
pub use common::*;

extern "C" {
    pub fn syscall(nb: Syscall, arg1: u32, arg2: u32, arg3: u32, arg4:
u32) -> i32;
}

pub struct Stdout {}

impl Write for Stdout {
    fn write_str(&mut self, s: &str) -> Result<(), Error> {
        unsafe { syscall(Syscall::Puts, &String::new(s) as *const String
as u32,  0, 0, 0); }
        Ok(())
    }
}

pub fn write_fmt(args: Arguments) {
    Stdout {}.write_fmt(args).ok();
}

#[macro_export]
macro_rules! print {
    ($($arg:tt)*) => (write_fmt(format_args!($($arg)*)));
}

```

```
#[macro_export]
macro_rules! println {
    () => (print!("\n"));
    ($fmt:expr) => (print!(concat!($fmt, "\n")));
    ($fmt:expr, $($arg:tt)*) => (print!(concat!($fmt, "\n"), $($arg)*));
}

pub fn clear() {
    copy_scr(&[[Character::null();BUFFER_WIDTH];BUFFER_HEIGHT]);
    set_cursor(0, 0);
}

pub fn puts(s: &str) {
    unsafe {
        syscall(Syscall::Puts, String::new(s).as_ptr() as u32,  0, 0, 0);
    }
}

pub fn putc(byte: u8) {
    unsafe {
        syscall(Syscall::Putc, byte as u32,  0, 0, 0);
    }
}

pub fn exec(s: &str) -> i32 {
    unsafe {
        syscall(Syscall::Exec, String::new(s).as_ptr() as u32,  0, 0, 0)
    }
}

pub fn keypressed() -> i32 {
    unsafe {
        syscall(Syscall::Keypressed, 0, 0, 0, 0)
    }
}

pub fn getc() -> u8 {
    unsafe {
        syscall(Syscall::Getc, 0, 0, 0, 0) as u8
    }
}

pub fn file_stat(s: &str) -> Stat {
    let mut stat = Stat::null();
    unsafe {
        syscall(Syscall::FileStat, String::new(s).as_ptr() as u32,
stat.as_ptr() as u32, 0, 0);
    }
```

```
75      return stat;
76  }
77
78  pub fn file_open(s: &str) -> i32 {
79      unsafe {
80          syscall(Syscall::FileOpen, String::new(s).as_ptr() as u32, 0, 0,
    0)
81      }
82  }
83
84  pub fn file_close(fd: u32) -> i32 {
85      unsafe {
86          syscall(Syscall::FileClose, fd, 0, 0, 0)
87      }
88  }
89
90  pub fn file_read(fd: u32, buf: *mut u8, n: u32) -> i32 {
91      unsafe {
92          syscall(Syscall::FileRead, fd, buf as u32, n, 0)
93      }
94  }
95
96  pub fn file_seek(fd: u32, offset: u32) -> i32 {
97      unsafe {
98          syscall(Syscall::FileSeek, fd, offset, 0, 0)
99      }
100 }
101
102 pub fn file_iterator() -> FileIterator {
103     let mut it = FileIterator::null();
104     unsafe {
105         syscall(Syscall::FileIterator, it.as_ptr() as u32, 0, 0, 0);
106     }
107     return it;
108 }
109
110 pub fn file_next(bytes: *const u8, it: *const FileIterator) -> i32 {
111     unsafe {
112         syscall(Syscall::FileNext, bytes as u32, it as u32, 0, 0)
113     }
114 }
115
116 pub fn get_ticks() -> u32 {
117     unsafe {
118         syscall(Syscall::GetTicks, 0, 0, 0, 0) as u32
119     }
120 }
121
```

```rust
pub fn sleep(ms: u32) {
    unsafe {
        syscall(Syscall::Sleep, ms, 0, 0, 0);
    }
}

pub fn set_cursor(x: u32, y: u32) {
    unsafe {
        syscall(Syscall::SetCursor, x, y, 0, 0);
    }
}

pub fn get_cursor(x: *const u32, y: *const u32) {
    unsafe {
        syscall(Syscall::GetCursor, x as u32, y as u32, 0, 0);
    }
}

pub fn cursor_disable(cd: bool) {
    unsafe {
        if cd {
            syscall(Syscall::CursorDisable, 1, 0, 0, 0);
        } else {
            syscall(Syscall::CursorDisable, 0, 0, 0, 0);
        }
    }
}

pub fn copy_scr(scr: *const FrameBuffer) {
    unsafe {
        syscall(Syscall::CopyScr, scr as u32, 0, 0, 0);
    }
}
```

VII.X /user/ulibc/src/mem.rs

```rust
// extern crate alloc;
use core::mem::size_of;
use rlibc::{memset,memcpy};
use io::*;

const FRAME_SIZE: usize = 0x1000;
const HEAP_END: u32 = 0xffffffff;

static mut HEAP_START: u32 = 0;
static mut HEAP_SIZE: usize = 0;

#[derive(Debug, Clone, Copy)]
```

```
#[repr(C, align(16))]
struct Header {
    previous: u32,
    next: u32,
    size: usize,
    free: bool
}

macro_rules! align {
    ($size:expr) => {
        if ($size & 0xfffffff0) != $size {
            ($size & 0xfffffff0) + 0x10
        } else {
            $size
        };
    }
}

fn heap_init() {
    unsafe {
        if HEAP_START == 0 {
            HEAP_START = syscall(Syscall::AllocFrame, 0, 0, 0, 0) as u32;
            HEAP_SIZE = (HEAP_END - HEAP_START) as usize;
            memset(HEAP_START as *mut u8, 0, FRAME_SIZE);
            memcpy(HEAP_START as *mut u8, Header::null(0,
HEAP_SIZE).as_ptr(), size_of::<Header>());
        }
    }
}

pub fn malloc(size: usize) -> u32 {
    heap_init();
    let aligned_size = align!(size);
    let mut addr = empty_block(aligned_size);
    let mut block = Header::from_ptr(addr as *mut u8);
    if block.size >= aligned_size && block.free {
        if block.next == 0 {
            block.insert_tail(addr, aligned_size);
        } else {
            block.insert(addr, aligned_size);
        }
        addr += size_of::<Header>() as u32;
        unsafe { memset(addr as *mut u8, 0, aligned_size); }
        return addr;
    }
    return 0;
}

```

```rust
pub fn free(addr: u32) {
    unsafe {
        let mut header_addr = addr - size_of::<Header>() as u32;
        let mut header = Header::from_ptr(header_addr as *const u8);
        if !header.free {
            let start = header_addr;
            let mut end = header.next;
            if header.previous != 0 {
                let previous = Header::from_ptr(header.previous as *const u8);
                if previous.free {
                    header_addr = header.previous;
                    header.previous = previous.previous;
                    header.size += previous.size + size_of::<Header>();
                }
            }
            if header.next != 0 {
               let mut next = Header::from_ptr(header.next as *const u8);
                if next.free {
                    header.next = next.next;
                    header.size += next.size + size_of::<Header>();
                }
                if header.next != 0 {
                    end = header.next;
                    next = Header::from_ptr(header.next as *const u8);
                    if next.previous != header_addr {
                        next.previous = header_addr;
                        memcpy(header.next as *mut u8, next.as_ptr(), size_of::<Header>());
                    }
                }
            }
            header.free = true;
            memcpy(header_addr as *mut u8, header.as_ptr(), size_of::<Header>());
            // free unused page tables
            let start_idx = start as usize / FRAME_SIZE;
            let mut end_idx = end as usize / FRAME_SIZE;
            if header.next == 0 {
                end_idx += 1;
            }
            for i in start_idx..end_idx {
                syscall(Syscall::FreeFrame, (i * FRAME_SIZE) as u32, 0, 0, 0);
            }
        }
    }
}
```

```
104
105 pub fn print_kmalloc_list() {
106     let mut addr = unsafe { HEAP_START };
107     while addr != 0 {
108         let block = Header::from_ptr(addr as *mut u8);
109         addr = block.next;
110         println!("{:x?}", block);
111     }
112     println!();
113 }
114
115 fn empty_block(size: usize) -> u32 {
116     let mut addr = unsafe { HEAP_START };
117     let mut block = Header::from_ptr(addr as *mut u8);
118     while block.next != 0 {
119         if block.size >= size && block.free {
120             break;
121         }
122         addr = block.next;
123         block = Header::from_ptr(addr as *mut u8);
124     }
125     return addr;
126 }
127
128 impl Header {
129     fn null(previous: u32, size: usize) -> Header {
130         Header {
131             previous: previous,
132             next: 0,
133             size: size,
134             free: true
135         }
136     }
137
138     fn insert(&mut self, addr: u32, size: usize) {
139         unsafe {
140             let total_size = size + size_of::<Header>();
141             if (addr as usize % FRAME_SIZE) + total_size > FRAME_SIZE {
142                 for _i in 1..(total_size / FRAME_SIZE + 1) {
143                     let entry_addr = syscall(Syscall::AllocFrame, 0, 0,
    0, 0) as u32;
144                     memset(entry_addr as *mut u8, 0, FRAME_SIZE);
145                 }
146             }
147             self.free = false;
148             if size == self.size {
149                 memcpy(addr as *mut u8, self.as_ptr(),
    size_of::<Header>());
```

```
150             } else {
151                 let tmp = self.next;
152                 let mut tmp_header = Header::from_ptr(tmp as *const u8);
153 
154                 self.size = size;
155                 self.next = addr + total_size as u32;
156                 let next_block_size = (tmp - self.next) as usize -
    size_of::<Header>();
157                 let mut next_header = Header::null(addr, next_block_size);
158                 next_header.next = tmp;
159                 tmp_header.previous = self.next;
160                 memcpy(addr as *mut u8, self.as_ptr(),
    size_of::<Header>());
161                 memcpy(self.next as *mut u8, next_header.as_ptr(),
    size_of::<Header>());
162                 memcpy(tmp as *mut u8, tmp_header.as_ptr(),
    size_of::<Header>());
163             }
164         }
165     }
166 
167     fn insert_tail(&mut self, addr: u32, size: usize) {
168         unsafe {
169             let total_size = size + size_of::<Header>();
170             self.size = size;
171             self.free = false;
172             self.next = addr + total_size as u32;
173             // alloc new frames if need
174             if (addr as usize % FRAME_SIZE) + total_size > FRAME_SIZE {
175                 for _i in 0..(total_size / FRAME_SIZE) {
176                     let entry_addr = syscall(Syscall::AllocFrame, 0, 0,
    0, 0) as u32;
177                     memset(entry_addr as *mut u8, 0, FRAME_SIZE);
178                 }
179             }
180             let tail_size = (HEAP_END - self.next) as usize -
    size_of::<Header>();
181             let mut tail = Header::null(addr, tail_size);
182             memcpy(addr as *mut u8, self.as_ptr(), size_of::<Header>());
183             memcpy(self.next as *mut u8, tail.as_ptr(),
    size_of::<Header>());
184         }
185     }
186 
187     fn as_ptr(&mut self) -> *const u8 {
188         self as *const Header as *const u8
189     }
190 
```

```
    fn from_ptr(ptr: *const u8) -> Header {
        unsafe {
            *(ptr as *const Header)
        }
    }
}
```

VII.XI /user/ulibc/src/ulibc.rs

```
#![feature(lang_items)]
// #![feature(alloc, global_allocator, allocator_api)]
#![no_std]

extern crate common;
pub use common::*;

extern crate rlibc;
pub use rlibc::*;

pub mod io;
pub mod curses;
pub mod mem;

#[lang = "panic_fmt"]
#[no_mangle]
pub extern "C" fn panic_fmt() -> ! {
    loop{}
}

#[no_mangle]
pub extern "C" fn __floatundisf() {
    loop {}
}
```

VII.XII /user/ulibc/Cargo.toml

```
[package]
name = "ulibc"
version = "0.1.0"
authors = ["orpheeantoniadis <orphee.antoniadis@gmail.com>"]

[lib]
name = "ulibc"
path = "src/ulibc.rs"

[dependencies]
rlibc = "1.0"
```

```
common = { path = "../../common" }
```

VII.XIII /user/app.ld

```
OUTPUT_FORMAT("binary")

SECTIONS {
    . = 0x0;                /* first section located at 0 */

    .entrypoint ALIGN(4):   /* entry point: must be located at 0 */
    {
        *(.entrypoint)
    }

    .text ALIGN(4) :        /* code */
    {
        *(.text*)
    }

    .rodata ALIGN(4) :      /* read-only data */
    {
        *(.rodata*)
    }

    .data ALIGN(4) :        /* initialized data */
    {
        *(.data*)
    }

    .bss ALIGN(4) :         /* unitialized data */
    {
        *(COMMON)
        *(.bss*)
    }
}
```

VII.XIV /user/entrypoint_asm.s

```
extern main

section .entrypoint
align 4

    mov     [stack_ptr],esp
    call    main
    mov     esp,[stack_ptr]
    iret
```

```
10
11 section .text
12 align 4
13
14 section .bss
15 stack_ptr:
16     resd 1    ; uninitialized dword (32 bits)
```

VII.XV /user/i386-app.json

```
1  {
2      "relocation_model": "static",
3      "data-layout": "e-m:e-p:32:32-f64:32:64-f80:32-n8:16:32-S128",
4      "llvm-target": "i386-unknown-none",
5      "target-endian": "little",
6      "target-pointer-width": "32",
7      "target-c-int-width": "32",
8      "os": "none",
9      "arch": "x86",
10     "linker-flavor": "gcc",
11     "features": "-mmx,-sse,+soft-float",
12     "disable-redzone": true,
13     "panic-strategy": "abort"
14 }
```

VII.XVI /user/Makefile

```
1  ARCH = i386
2  TARGET = $(ARCH)-app
3  BUILD_FOLDER = build
4
5  LINKER = app.ld
6  FLAGS = -T $(LINKER) -m32 -MMD -g -ffreestanding -nostdlib -Wall -Wextra
   -fno-pie
7
8  APPS = hello demo shell splash
9
10 SRCS = $(wildcard *.s)
11 OBJS = $(patsubst %.s, $(BUILD_FOLDER)/%.o, $(SRCS))
12 ROBJS = $(foreach A,$(APPS),$A/target/$(TARGET)/release/lib$A.a)
13 EXECS = $(patsubst %, $(BUILD_FOLDER)/%, $(APPS))
14
15 all : build
16
17 build : $(EXECS)
18
19 $(EXECS) : $(ROBJS)
```

```
20
21  %.a : $(OBJS)
22      $(MAKE) -C $(shell echo "$@" | cut -d "/" -f1)
23      gcc $(FLAGS) $^ $@ -o $(BUILD_FOLDER)/$(shell echo "$@" | cut -d "/"
    -f1)
24
25  $(BUILD_FOLDER)/%.o : %.s
26      $(shell mkdir -p $(BUILD_FOLDER))
27      nasm -f elf32 $< -o $@
28
29  clean :
30      $(MAKE) -C hello clean
31      $(MAKE) -C demo clean
32      $(MAKE) -C shell clean
33      rm -rf build
34
35  rebuild : clean build
36
```

VII.XVII /user/syscall_asm.s

```
1   global syscall
2
3   section .text                       ; start of the text (code) section
4   align 4                             ; the code must be 4 byte aligned
5
6   ; int syscall(uint32_t nb, uint32_t arg1, uint32_t arg2, uint32_t arg3,
    uint32_t arg4);
7   syscall:
8       ; parameters cannot be passed into the stack because the
    trap/interrupt gate
9       ; performs a stack switch (from user stack to kernel stack). By the
    time we're
10      ; in the syscall handler we're accessing the kernel stack
    (tss.ss/tss.esp).
11      push    ebp
12      mov     ebp,esp
13
14      ; save all general registers since we modify them below
15      ; eax is not saved as it's used to store the syscall's return value
16      push    ebx
17      push    ecx
18      push    edx
19      push    esi
20      push    edi
21
22      mov     eax,[ebp+8]
23      mov     ebx,[ebp+12]
```

```
24      mov     ecx,[ebp+16]
25      mov     edx,[ebp+20]
26      mov     esi,[ebp+24]
27      int     48
28
29      ; restore all general registers
30      pop     edi
31      pop     esi
32      pop     edx
33      pop     ecx
34      pop     ebx
35
36      mov     esp,ebp
37      pop     ebp
38      ret
```

VIII /Makefile

```
KERNEL_PATH = kernel/

.PHONY: all build run test doc clean mrproper

all: build

build:
    $(MAKE) -C $(KERNEL_PATH)

run:
    $(MAKE) -C $(KERNEL_PATH) run

test:
    $(MAKE) -C $(KERNEL_PATH) test

doc:
    $(MAKE) -C $(KERNEL_PATH) doc

clean:
    $(MAKE) -C $(KERNEL_PATH) clean

mrproper:
    $(MAKE) -C $(KERNEL_PATH) mrproper
```