

Annexe A :

Code source

Orphée Antoniadis

Thèse de Bachelor - Prof. Florent Glück

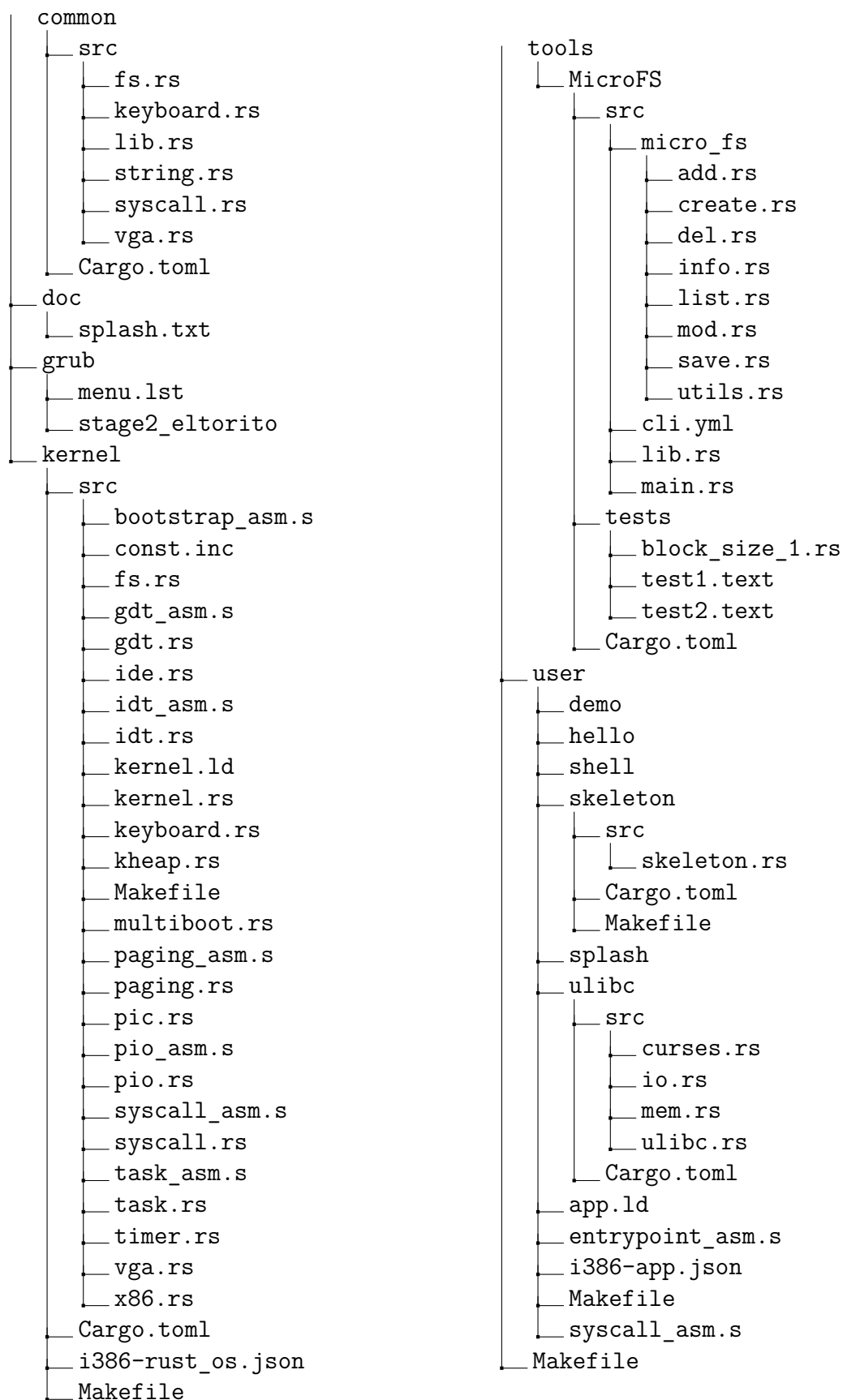
hepia ITI Septembre 2018

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/MicroFS/src/micro_fs/add.rs	83
VI.II	/tools/MicroFS/src/micro_fs/create.rs	83
VI.III	/tools/MicroFS/src/micro_fs/del.rs	85
VI.IV	/tools/MicroFS/src/micro_fs/info.rs	85
VI.V	/tools/MicroFS/src/micro_fs/list.rs	85
VI.VI	/tools/MicroFS/src/micro_fs/mod.rs	86
VI.VII	/tools/MicroFS/src/micro_fs/save.rs	89
VI.VIII	/tools/MicroFS/src/micro_fs/utils.rs	90
VI.IX	/tools/MicroFS/src/cli.yml	93
VI.X	/tools/MicroFS/src/lib.rs	94
VI.XI	/tools/MicroFS/src/main.rs	95
VI.XII	/tools/MicroFS/tests/block_size_1.rs	97
VI.XIII	/tools/MicroFS/Cargo.toml	109
VII	/user	110
VII.I	/user/demo/src/demo.rs	110
VII.II	/user/hello/src/hello.rs	111
VII.III	/user/shell/src/shell.rs	112
VII.IV	/user/skeleton/src/skeleton.rs	114
VII.V	/user/skeleton/Cargo.toml	114
VII.VI	/user/skeleton/Makefile	115
VII.VII	/user/splash/src/splash.rs	115
VII.VIII	/user/ulibc/src/curses.rs	116
VII.IX	/user/ulibc/src/io.rs	117
VII.X	/user/ulibc/src/mem.rs	121
VII.XI	/user/ulibc/src/ulibc.rs	125
VII.XII	/user/ulibc/Cargo.toml	126
VII.XIII	/user/app.ld	126
VII.XIV	/user/entrypoint_asm.s	127
VII.XV	/user/i386-app.json	127
VII.XVI	/user/Makefile	128
VII.XVII	/user/syscall_asm.s	128
VIII	/Makefile	130

I Structure du projet



II /common

II.I /common/src/fs.rs

```
1 pub const MAX_FILENAME_LENGTH: usize = 26;
2
3 #[derive(Debug, Clone, Copy)]
4 #[repr(C)]
5 pub struct Stat {
6     pub name: [u8; MAX_FILENAME_LENGTH],
7     pub size: usize,
8     pub entry_offset: u16,
9     pub start: usize
10 }
11
12 #[derive(Debug, Clone, Copy)]
13 #[repr(C)]
14 pub struct FileIterator {
15     pub sector: u32,
16     pub offset: usize
17 }
18
19 impl Stat {
20     pub fn null() -> Stat {
21         Stat {
22             name: [0; MAX_FILENAME_LENGTH],
23             size: 0,
24             entry_offset: 0,
25             start: 0
26         }
27     }
28
29     pub fn as_ptr(&mut self) -> *const Stat {
30         self as *const Stat
31     }
32 }
33
34 impl FileIterator {
35     pub fn null() -> FileIterator {
36         FileIterator {
37             sector: 0,
38             offset: 0
39         }
40     }
41
42     pub fn as_ptr(&mut self) -> *const FileIterator {
43         self as *const FileIterator
44     }
45 }
```

II.II /common/src/keyboard.rs

```

1  // Ascii charset hex
2  pub const NUL: char = '\0';
3  pub const BACKSPACE: char = 0x8 as char;
4  pub const NAK: char = 0x15 as char;
5  pub const E_ACUTE: char = 0x82 as char;
6  pub const A_GRAVE: char = 0x85 as char;
7  pub const C_CEDILLA: char = 0x87 as char;
8  pub const E_GRAVE: char = 0x8a as char;
9  pub const U_GRAVE: char = 0x97 as char;
10 pub const POUND: char = 0x9c as char;
11
12 // Keys codes
13 pub const CTRL: u8 = 29;
14 pub const LEFT_SHIFT: u8 = 42;
15 pub const RIGHT_SHIFT: u8 = 54;
16 pub const ALT: u8 = 56;
17 pub const CAPS_LOCK: u8 = 58;
18 pub const UP_KEY: u8 = 72;
19 pub const LEFT_KEY: u8 = 75;
20 pub const RIGHT_KEY: u8 = 77;
21 pub const DOWN_KEY: u8 = 80;
22 pub const LEFT_CMD: u8 = 91;
23 pub const RIGHT_CMD: u8 = 92;
24
25 pub const KEY_MAP: [char;93] = [
26     NUL, NUL, '&', E_ACUTE, '"', '\\', '(', NAK, E_GRAVE, '!', C_CEDILLA,
27     A_GRAVE, ')', '-', BACKSPACE,
28     '\t', 'a', 'z', 'e', 'r', 't', 'y', 'u', 'i', 'o', 'p', '^', '$',
29     '\n',
30     NUL, 'q', 's', 'd', 'f', 'g', 'h', 'j', 'k', 'l', 'm', U_GRAVE, '@',
31     NUL, '`', 'w', 'x', 'c', 'v', 'b', 'n', ',', ';', ':', '=', NUL,
32     NUL, NUL, ' ', NUL, ' ', NUL, NUL, NUL, NUL, NUL, NUL, NUL, NUL,
33     NUL, NUL, NUL, NUL, NUL, NUL, NUL, NUL, NUL, NUL, NUL, NUL, NUL,
34     NUL, NUL, NUL, NUL, NUL, '<', NUL, NUL, NUL, NUL, NUL, NUL
35 ];
36
37 pub const SHIFT_KEY_MAP: [char;93] = [
38     NUL, NUL, '1', '2', '3', '4', '5', '6', '7', '8', '9', '0', '°', '_',
39     BACKSPACE,
40     '\t', 'A', 'Z', 'E', 'R', 'T', 'Y', 'U', 'I', 'O', 'P', NUL, '*',
41     '\n',
42     NUL, 'Q', 'S', 'D', 'F', 'G', 'H', 'J', 'K', 'L', 'M', '%', '#',
43     NUL, POUND, 'W', 'X', 'C', 'V', 'B', 'N', '?', '.', '/', '+', NUL,
44     NUL, NUL, ' ', NUL, ' ', NUL, NUL, NUL, NUL, NUL, NUL, NUL, NUL,

```

```

41     NUL, NUL, NUL, NUL, NUL, NUL, NUL, NUL, NUL, NUL, NUL, NUL, NUL,
42     NUL, NUL, NUL, NUL, NUL, '>', NUL, NUL, NUL, NUL, NUL, NUL
43 ];

```

II.III /common/src/lib.rs

```

1  #![feature(const_fn)]
2  #![no_std]
3
4  mod syscall;
5  mod string;
6  mod fs;
7  mod keyboard;
8  mod vga;
9
10 pub use syscall::*;
11 pub use string::*;
12 pub use fs::*;
13 pub use keyboard::*;
14 pub use vga::*;

```

II.IV /common/src/string.rs

```

1  use core::slice::from_raw_parts;
2  use core::str::from_utf8;
3
4  pub const MAX_STR_LEN : usize = 256;
5
6  #[derive(Debug, Clone, Copy)]
7  #[repr(C)]
8  pub struct String {
9      pub bytes_ptr: u32,
10     pub len: usize
11 }
12
13 pub fn bytes_to_str(bytes: &[u8]) -> &str {
14     let mut cnt = 0;
15     for &byte in bytes {
16         if byte == 0 {
17             break;
18         } else if byte > 0x7f {
19             return "\0";
20         }
21         cnt += 1;
22     }
23     from_utf8(&bytes[0..cnt]).expect("Found invalid UTF-8")
24 }

```



```

25
26 impl String {
27     pub fn new(s: &str) -> String {
28         unsafe {
29             String {
30                 bytes_ptr: &from_raw_parts(s.as_ptr(), s.len())[0] as
*const u8 as u32,
31                 len: s.len()
32             }
33         }
34     }
35
36     pub fn to_string(&mut self) -> &str {
37         unsafe {
38             let addr = self.bytes_ptr as *const u8;
39             let bytes = &*(addr as *const [u8; MAX_STR_LEN]);
40             from_utf8(&bytes[0..self.len]).expect("Found invalid UTF-8")
41         }
42     }
43
44     pub fn offset(&mut self, offset: u32) {
45         self.bytes_ptr += offset;
46     }
47
48     pub fn as_ptr(&mut self) -> *const String {
49         self as *const String
50     }
51 }

```

II.V /common/src/syscall.rs

```

1  #[repr(u8)]
2  pub enum Syscall {
3      Puts          = 0x00,
4      Putc          = 0x01,
5      Exec          = 0x02,
6      Keypressed    = 0x03,
7      Getc          = 0x04,
8      FileStat      = 0x05,
9      FileOpen      = 0x06,
10     FileClose     = 0x07,
11     FileRead      = 0x08,
12     FileSeek      = 0x09,
13     FileIterator   = 0x0a,
14     FileNext       = 0x0b,
15     GetTicks       = 0x0c,
16     Sleep          = 0x0d,
17     SetCursor      = 0x0e,

```

```

18     GetCursor      = 0x0f,
19     CursorDisable  = 0x10,
20     CopyScr        = 0x11,
21     AllocFrame     = 0x12,
22     FreeFrame      = 0x13
23 }

```

II.VI /common/src/vga.rs

```

1  pub const BUFFER_HEIGHT : usize = 25;
2  pub const BUFFER_WIDTH  : usize = 80;
3  pub const FG_COLOR      : Color = Color::Black;
4  pub const BG_COLOR      : Color = Color::White;
5
6  #[repr(u8)]
7  #[derive(Copy, Clone)]
8  pub enum Color {
9      Black      = 0x0,
10     Blue       = 0x1,
11     Green      = 0x2,
12     Cyan       = 0x3,
13     Red        = 0x4,
14     Magenta    = 0x5,
15     Brown      = 0x6,
16     LightGray  = 0x7,
17     DarkGray   = 0x8,
18     LightBlue  = 0x9,
19     LightGreen = 0xa,
20     LightCyan  = 0xb,
21     LightRed   = 0xc,
22     Pink       = 0xd,
23     Yellow     = 0xe,
24     White      = 0xf,
25 }
26
27 #[derive(Debug, Clone, Copy)]
28 pub struct ColorAttribute(u8);
29
30 #[derive(Debug, Clone, Copy)]
31 #[repr(C)]
32 pub struct Character {
33     pub ascii: u8,
34     pub attribute: ColorAttribute,
35 }
36
37 pub type FrameBuffer = [[Character; BUFFER_WIDTH]; BUFFER_HEIGHT];
38
39 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) ->
86     ColorAttribute {
87         ColorAttribute((background as u8) << 4 | (foreground as u8))

```

```

87     }
88 }
89
90 impl Character {
91     pub const fn null() -> Character {
92         Character {
93             ascii: 0,
94             attribute: ColorAttribute::new(BG_COLOR, FG_COLOR)
95         }
96     }
97
98     pub const fn new(ascii: u8, attribute: ColorAttribute) -> Character {
99         Character {
100             ascii: ascii,
101             attribute: attribute
102         }
103     }
104 }

```

II.VII /common/Cargo.toml

```

1 [package]
2 name = "common"
3 version = "0.1.0"
4 authors = ["orpheeantoniadis <orphee.antoniadis@gmail.com>"]
5
6 [dependencies]

```

III /doc/splash.txt

```
1      _ _ _ _ _  
2  /  _ _ \  _ _ _ _ _ /  /  /  _ _ \  _ _ _ _ _  
3  /  /  /  /  /  /  /  /  _ _ _ _ _ /  /  /  /  /  \  _ _ _ _ _  
4  /  _ ,  _ /  /  /  (  _ ) /  /  /  /  /  _ _ _ _ _ /  
5  /  /  |  | \  _ ,  /  _ _ _ _ _ \  _ _ _ _ _ \  _ _ _ _ _ /  
6  _ /  _ /  _ _ _ _ _
```

IV /grub/menu.lst

```
1 default=0
2 timeout=3
3 title RustOS
4 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
34 http://git.savannah.gnu.org/cgit/grub.git/tree/doc/multiboot.h?h=multiboot
35 dd MULTIBOOT_MAGIC
36 dd MULTIBOOT_FLAGS
37 dd MULTIBOOT_CHECKSUM
38
39 ;-----
40 section .low_text
41
42 entrypoint:
43     ; save multiboot infos
44     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 {

```

```

165         return cnt;
166     }
167     cnt += 1;
168 }
169 return -1;
170 }
171 }
172
173 impl FdtEntry {
174     const fn null() -> FdtEntry {
175         FdtEntry {
176             stat: Stat {
177                 name: [0;MAX_FILENAME_LENGTH],
178                 size: 0,
179                 entry_offset: 0,
180                 start: 0
181             },
182             pos: 0
183         }
184     }
185 }
186
187 impl StatBuilder for Stat {
188     fn new(filename: &str) -> Stat {
189         let mut sector : [u16;SECTOR_SIZE/2] = [0;SECTOR_SIZE/2];
190         let mut raw_filename = [0;MAX_FILENAME_LENGTH];
191         let mut it = FileIterator::new();
192         while it.has_next() {
193             it.next(&mut raw_filename[0]);
194             if filename == bytes_to_str(&raw_filename) {
195                 read_sector(it.sector, &mut sector[0] as *mut u16);
196                 unsafe {
197                     let offset = it.offset - ENTRY_SIZE;
198                     let entries = mem::transmute:::<[u16;SECTOR_SIZE/2],
199 [u8;SECTOR_SIZE]>(sector);
200                     let start = [entries[offset+26], entries[offset+27]];
201                     let start = mem::transmute:::<[u8;2], u16>(start);
202                     let size = [entries[offset+28], entries[offset+29],
203 entries[offset+30], entries[offset+31]];
204                     let size = mem::transmute:::<[u8;4], u32>(size);
205                     return Stat {
206                         name: raw_filename,
207                         size: size as usize,
208                         entry_offset: offset as u16,
209                         start: start as usize
210                     }
211                 }
212             }
213         }
214     }
215 }

```

```

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 } /
220 SECTOR_SIZE) as u32,
221             offset: 0
222         }
223     }
224
225     fn has_next(&mut self) -> bool {
226         if self.sector < self.sector + unsafe { SB.block_size /
227 SECTOR_SIZE } as u32 {
228             let mut sector : [u16;SECTOR_SIZE/2] = [0;SECTOR_SIZE/2];
229             read_sector(self.sector, &mut sector[0] as *mut u16);
230             let entries = unsafe {
231                 mem::transmute::<[u16;SECTOR_SIZE/2],
232 [u8;SECTOR_SIZE]>(sector)
233             };
234             if entries[self.offset] != 0 {
235                 return true;
236             }
237         }
238         return false;
239     }
240
241     fn next(&mut self, filename: *mut u8) -> i8 {
242         unsafe {
243             if self.has_next() {
244                 let mut sector : [u16;SECTOR_SIZE/2] = [0;SECTOR_SIZE/2];
245                 read_sector(self.sector, &mut sector[0] as *mut u16);
246                 let entries = mem::transmute::<[u16;SECTOR_SIZE/2],
247 [u8;SECTOR_SIZE]>(sector);
248                 memcpy(filename, &entries[self.offset],
249 MAX_FILENAME_LENGTH);
250                 self.offset = (self.offset + ENTRY_SIZE) % SECTOR_SIZE;
251                 if self.offset == 0 {
252                     self.sector += 1;
253                 }
254                 return 0;
255             }
256             return -1;
257         }
258     }
259 }

```

```

254 }
255
256 impl Superblock {
257     const fn null() -> Superblock {
258         Superblock { block_size: 0, fat_size: 0, root_entry: 0 }
259     }
260
261     fn new() -> Superblock {
262         let mut sector : [u16;SECTOR_SIZE/2] = [0;SECTOR_SIZE/2];
263         read_sector(0, &mut sector[0] as *mut u16);
264         let raw_sb = unsafe {
265             mem::transmute:::<[u16;SECTOR_SIZE/2],
266             [u8;SECTOR_SIZE]>(sector)
267         };
268         let label = bytes_to_str(&raw_sb[0x52..0x59]);
269         let block_size = raw_sb[13] as usize * SECTOR_SIZE;
270         let fat_size = unsafe {
271             mem::transmute:::<[u8;4], u32>([raw_sb[0x24], raw_sb[0x25],
272             raw_sb[0x26], raw_sb[0x27]])
273         };
274         let root_entry = raw_sb[0x2c];
275         println!("\n{} ready.", label);
276         println!("Block size = {} bytes", block_size);
277         println!("FAT size = {} bytes", fat_size);
278         println!("Root entry = block number {}\n", root_entry);
279
280         Superblock {
281             block_size: block_size,
282             fat_size: fat_size as usize,
283             root_entry: root_entry as usize
284         }
285     }
286 }

```

V.IV /kernel/src/gdt_asm.s

```

1  %include "const.inc"
2
3  global gdt_load
4
5  section .text:                ; start of the text (code) section
6
7  gdt_load:
8      mov     eax,[esp+4]      ; Get the pointer to the GDT, passed as a
9      lgdt    [eax]           ; Load the new GDT pointer
10     mov     ax,GDT_KERNEL_DATA_SELECTOR ; offset in the GDT of the
kernel data segment

```



```

11     mov     ds,ax           ; Load all data segment selectors
12     mov     es,ax
13     mov     fs,ax
14     mov     gs,ax
15     mov     ss,ax
16     jmp     GDT_KERNEL_CODE_SELECTOR:.flush ; far jump [selector:offset]
17     .flush:
18     ret

```

V.V /kernel/src/gdt.rs

```

1  ///! Module for the memory management of RustOS using a Global
2  Descriptor Table
3  #![allow(dead_code)]
4
5  use core::mem::size_of;
6  use x86::*;
7  use task::*;
8
9  /// The GDT size (not including the tss and ldt entries)
10 pub const GDT_SIZE: usize = 6;
11
12 /// Converts a descriptor index in the GDT into a selector
13 pub const fn gdt_index_to_selector(idx: u32) -> u32 {idx << 3}
14 /// Converts a descriptor selector in the GDT into an index
15 pub const fn selector_to_gdt_index(idx: u32) -> u32 {idx >> 3}
16
17 /// The Global Descriptor Table of RustOS
18 pub static mut GDT: Gdt = [GdtEntry::null();GDT_SIZE+TASKS_NB];
19 static mut GDT_PTR: GdtPtr = GdtPtr::null();
20
21 /// Defines a Global Descriptor Table
22 pub type Gdt = [GdtEntry; GDT_SIZE+TASKS_NB];
23
24 /// Structure of a GDT descriptor. There are 2 types of descriptors:
25 segments and TSS.
26 /// Section 3.4.5 of Intel 64 & IA32 architectures software developer's
27 manual describes
28 /// segment descriptors while section 6.2.2 describes TSS descriptors.
29 #[derive(Debug, Clone, Copy)]
30 #[repr(C, packed)]
31 pub struct GdtEntry {
32     lim15_0: u16,
33     base15_0: u16,
34     base23_16: u8,
35
36     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
57         // segment.
58         // Code and data segments must have a privilege level of 0.
59         GDT[0] = GdtEntry::null();
60         GDT[1] = GdtEntry::make_code_segment(0, 0xffff, DPL_KERNEL);
61         GDT[2] = GdtEntry::make_data_segment(0, 0xffff, DPL_KERNEL);
62         GDT[3] = GdtEntry::make_code_segment(0, 0xffff, DPL_USER);
63         GDT[4] = GdtEntry::make_data_segment(0, 0xffff, DPL_USER);
64         // setup gdt_ptr so it points to the GDT and ensure it has the
65         // right limit.
66         GDT_PTR = GdtPtr::new((size_of::<Gdt>() - 1) as u16, &GDT);
67         // Load the GDT
68         gdt_load(&GDT_PTR);
69         // Init tasks
70         tasks_init();
71     }
72 }
73
74 impl GdtEntry {
75     /// Create a null segment
76     pub const fn null() -> GdtEntry {
77         GdtEntry {
78             lim15_0: 0,
79             base15_0: 0,
80             base23_16: 0,
81             flags7_0: 0,
82             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_EXECPREAD,
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 {

```

```

119         ((self as *mut _ as u32) - (&GDT as *const _ as u32)) >> 3
120     }
121 }
122
123 /// Converts a GDT entry to its selector in the GDT
124 pub fn to_selector(&mut self) -> u32 {
125     unsafe {
126         (self as *mut _ as u32) - (&GDT as *const _ as u32)
127     }
128 }
129 }
130
131 impl GdtPtr {
132     const fn null() -> GdtPtr {
133         GdtPtr {
134             limit: 0,
135             base: 0 as *const _
136         }
137     }
138
139     fn new(limit: u16, base: *const Gdt) -> GdtPtr {
140         GdtPtr {
141             limit: limit,
142             base: base
143         }
144     }
145 }

```

V.VI /kernel/src/ide.rs

```

1  #![allow(dead_code)]
2
3  /**
4   * Simple IDE read/write routines using PIO mode.
5   * This code is very CPU intensive and not efficient
6   * (if that's what you're after, use DMA mode instead).
7   * Reference: http://wiki.osdev.org/ATA\_PIO\_Mode
8   * ATA disk0, I/O ports: 0x1f0-0x1f7, 0x3f6
9   * ATA disk1, I/O ports: 0x170-0x177, 0x376
10  */
11
12 use pio::*;
13
14 // IDE ports
15 const IDE_CMD : u16 = 0x1f7;
16 const IDE_DATA : u16 = 0x1f0;
17
18 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
29  mode.
30  * @param sector the sector to read or write (0-indexed).
31  */
32 fn pio_prepare(sector: u32) {
33     unsafe {
34         wait_drive();
35         outb(0x1f2, 1); // 1 sector
36         outb(0x1f3, (sector & 0xff) as u8); // send
37         bits 0-7 of LBA
38         outb(0x1f4, ((sector >> 8) & 0xff) as u8); // send
39         bits 8-15 of LBA
40         outb(0x1f5, ((sector >> 16) & 0xff) as u8); // send
41         bits 16-23 of LBA
42         outb(0x1f6, ((sector >> 24) & 0x0f) as u8 | 0xe0); // send
43         bits 24-27 of LBA + set LBA mode; 0xe0 = 11100000b;
44     }
45 }
46
47 /**
48  * Read sectors from the first disk.
49  * @param sector first sector to read (0-indexed)
50  * @param dst address to store to read data
51  * Based on the assembly code at
52  http://wiki.osdev.org/ATA\_read/write\_sectors
53  */
54 pub fn read_sector(sector: u32, dst: *mut u16) {
55     unsafe {
56         pio_prepare(sector);
57
58         outb(IDE_CMD, 0x20); // read with retry
59         wait_drive();
60
61         for i in 0..(SECTOR_SIZE/2) {
62             *dst.offset(i as isize) = inw(IDE_DATA);
63         }
64     }
65 }
66

```

```

61  /**
62   * Write sectors from the first disk.
63   * @param sector first sector to write (0-indexed)
64   * @param src address of the data to be written
65   */
66  pub fn write_sector(sector: u32, src: *mut u16) {
67      unsafe {
68          pio_prepare(sector);
69
70          outb(IDE_CMD, 0x30); // write with retry
71          wait_drive();
72
73          for i in 0..(SECTOR_SIZE/2) {
74              outw(IDE_DATA, *src.offset(i as isize));
75          }
76      }
77  }

```

V.VII /kernel/src/idt_asm.s

```

1  %include "const.inc"
2
3  extern exception_handler
4  extern irq_handler
5
6  section .text    ; start of the text (code) section
7  align 4          ; the code must be 4 byte aligned
8
9  ;-----
10 ; CPU exceptions
11 ; Macro to generate exceptions. The only argument is for exception's
   digit
12 %macro exception 1
13 global _exception_%1
14 _exception_%1:
15     cli                ; disable interrupts
16     ; this if is for exceptions without error code
17     %if %1 < 8 || %1 == 9 || %1 == 15 || %1 == 16 || %1 > 17
18     push    0          ; dummy error code in certain case
19     %endif
20     push    %1          ; exception number
21     jmp     exception_wrapper
22 %endmacro
23 ; Creation of all exceptions (0 to 20), total = 21
24 %assign i 0
25 %rep 21
26 exception i
27 %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

```

```

75     call    %1_handler ; implemented in idt.c
76     pop     eax ; only here to balance the "push eax" done before the
call
77
78     ; Restore all registers
79     pop     gs
80     pop     fs
81     pop     es
82     pop     ds
83     pop     ebp
84     pop     edi
85     pop     esi
86     pop     edx
87     pop     ecx
88     pop     ebx
89     pop     eax
90
91     ; Fix the stack pointer due to the 2 push done before the call to
92     ; exception_wrapper: error code and exception/irq number
93     add     esp,8
94     iret
95 %endmacro
96
97 wrapper exception
98 wrapper irq
99
100 ;-----
101 ; Load the IDT
102 global idt_load
103 ; Argument : address of idt structure
104 idt_load:
105     mov eax, [esp + 4]
106     lidt [eax]
107     ret

```

V.VIII /kernel/src/idt.rs

```

1  //! Module for the intrruption management of RustOS using an Interrupt
Descriptor Table
2  #![allow(dead_code)]
3
4  use core::mem::size_of;
5  use x86::*;
6  use vga::*;
7  use pic::*;
8  use timer::timer_handler;
9  use keyboard::keyboard_handler;
10 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.

```

```

57 // This format is required by the lgdt instruction.
58 #[derive(Debug, Clone, Copy)]
59 #[repr(C, packed)]
60 struct IdtPtr {
61     limit: u16, // Limit of the table (ie. its size)
62     base: *const Idt // Address of the first entry
63 }
64
65 /// CPU context used when saving/restoring context from an interrupt
66 #[derive(Debug, Clone, Copy)]
67 #[repr(C, packed)]
68 pub struct Regs {
69     gs: u32, fs: u32, es: u32, ds: u32,
70     ebp: u32, edi: u32, esi: u32,
71     edx: u32, ecx: u32, ebx: u32, eax: u32,
72     number: u32, error_code: u32,
73     eip: u32, cs: u32, eflags: u32, esp: u32, ss: u32
74 }
75
76 /// Initialize the Interrupt Descriptor Table
77 pub fn idt_init() {
78     unsafe {
79         // CPU exceptions
80         IDT[0] = IdtEntry::new(GDT_KERNEL_CODE_SELECTOR as u16,
81 _exception_0 as *const () as u32, TYPE_INTERRUPT_GATE, DPL_KERNEL);
82         IDT[1] = IdtEntry::new(GDT_KERNEL_CODE_SELECTOR as u16,
83 _exception_1 as *const () as u32, TYPE_INTERRUPT_GATE, DPL_KERNEL);
84         IDT[2] = IdtEntry::new(GDT_KERNEL_CODE_SELECTOR as u16,
85 _exception_2 as *const () as u32, TYPE_INTERRUPT_GATE, DPL_KERNEL);
86         IDT[3] = IdtEntry::new(GDT_KERNEL_CODE_SELECTOR as u16,
87 _exception_3 as *const () as u32, TYPE_INTERRUPT_GATE, DPL_KERNEL);
88         IDT[4] = IdtEntry::new(GDT_KERNEL_CODE_SELECTOR as u16,
89 _exception_4 as *const () as u32, TYPE_INTERRUPT_GATE, DPL_KERNEL);
90         IDT[5] = IdtEntry::new(GDT_KERNEL_CODE_SELECTOR as u16,
91 _exception_5 as *const () as u32, TYPE_INTERRUPT_GATE, DPL_KERNEL);
92         IDT[6] = IdtEntry::new(GDT_KERNEL_CODE_SELECTOR as u16,
93 _exception_6 as *const () as u32, TYPE_INTERRUPT_GATE, DPL_KERNEL);
94         IDT[7] = IdtEntry::new(GDT_KERNEL_CODE_SELECTOR as u16,
95 _exception_7 as *const () as u32, TYPE_INTERRUPT_GATE, DPL_KERNEL);
96         IDT[8] = IdtEntry::new(GDT_KERNEL_CODE_SELECTOR as u16,
97 _exception_8 as *const () as u32, TYPE_INTERRUPT_GATE, DPL_KERNEL);
98         IDT[9] = IdtEntry::new(GDT_KERNEL_CODE_SELECTOR as u16,
99 _exception_9 as *const () as u32, TYPE_INTERRUPT_GATE, DPL_KERNEL);
100        IDT[10] = IdtEntry::new(GDT_KERNEL_CODE_SELECTOR as u16,
101 _exception_10 as *const () as u32, TYPE_INTERRUPT_GATE, DPL_KERNEL);
102        IDT[11] = IdtEntry::new(GDT_KERNEL_CODE_SELECTOR as u16,
103 _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
167 0)
168 // type indicates the IDT entry type
169 // dpl is the privilege level required to call the associated ISR
170 fn new(selector: u16, offset: u32, idt_type: u8, dpl: u8) ->
171 IdtEntry {
172     IdtEntry {
173         offset15_0: (offset & 0xffff) as u16,
174         selector: selector,
175         reserved: 0,
176         flags: idt_type | dpl<<5 | 1<<7,
177         offset31_16: ((offset >> 16) & 0xffff) as u16
178     }
179 }
180
181 impl IdtPtr {
182     const fn null() -> IdtPtr {
183         IdtPtr {
184             limit: 0,
185             base: 0 as *const _
186         }
187     }
188
189     fn new(limit: u16, base: *const Idt) -> IdtPtr {
190         IdtPtr {
191             limit: limit,
192             base: base
193         }
194     }
195 }
196
197 extern "C" {
198     fn idt_load(idt_ptr: *const IdtPtr);
199
200     // Exception handler
201     fn _exception_0();
202     fn _exception_1();
203     fn _exception_2();
204     fn _exception_3();
205     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)

```

```

11     }
12     .low_text ALIGN (4K) :
13     {
14         *(.low_text)
15     }
16     .low_data ALIGN (4K) :
17     {
18         *(.low_data)
19     }
20     .low_bss ALIGN (4K) :
21     {
22         low_bss_start = .;
23         *(.low_bss)
24         low_bss_end = .;
25     }
26     low_kernel_end = .;
27
28     /* Higher-half Kernel */
29     . += 0xC0000000;
30     kernel_start = .;
31     /* kernel stack */
32     .stack ALIGN(4) : AT(ADDR(.stack) - 0xC0000000)
33     {
34         *(.stack)
35     }
36     /* code */
37     .text ALIGN(4K) : AT(ADDR(.text) - 0xC0000000)
38     {
39         *(.text*)
40     }
41     /* read-only data */
42     .rodata ALIGN(4K) : AT(ADDR(.rodata) - 0xC0000000)
43     {
44         *(.rodata*)
45     }
46     /* initialized data */
47     .data ALIGN(4K) : AT(ADDR(.data) - 0xC0000000)
48     {
49         *(.data*)
50     }
51     /* uninitialized data */
52     .bss ALIGN(4K) : AT(ADDR(.bss) - 0xC0000000)
53     {
54         bss_start = .;
55         *(COMMON)
56         *(.bss*)
57         bss_end = .;
58     }

```

```

59     kernel_end = .;
60
61     /* Utils */
62     low_kernel_size = low_kernel_end - low_kernel_start;
63     low_bss_size = low_bss_end - low_bss_start;
64     kernel_size = kernel_end - kernel_start;
65     bss_size = bss_end - bss_start;
66 }
67
68 ASSERT(kernel_size < 0x300000, "Kernel exceeds the 3 MB limit!");

```

V.X /kernel/src/kernel.rs

```

1  ///! # RustOS
2  ///!
3  ///! `rust_os` is a kernel running on IA-32 architecture
4
5  #![feature(lang_items, asm, const_fn)]
6  #![no_std]
7
8  extern crate rlibc;
9  extern crate common;
10
11 pub mod x86;
12 pub mod multiboot;
13 pub mod vga;
14 pub mod pio;
15 pub mod paging;
16 pub mod kheap;
17 pub mod gdt;
18 pub mod pic;
19 pub mod idt;
20 pub mod timer;
21 pub mod keyboard;
22 pub mod ide;
23 pub mod fs;
24 pub mod task;
25 pub mod syscall;
26
27 use x86::*;
28 use multiboot::*;
29 use vga::*;
30 use pio::disable_cursor;
31 use paging::*;
32 use kheap::*;
33 use gdt::gdt_init;
34 use pic::pic_init;
35 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))]

```

```

82  #[lang = "panic_fmt"]
83  #[no_mangle]
84  pub extern fn panic_fmt(details: ::core::fmt::Arguments, file: &'static
str, line: u32, column: u32) -> ! {
85      println!("panicked at {}, {}:{}:{}", details, file, line, column);
86      loop {
87          cli();
88          halt();
89      }
90  }
91
92  #[no_mangle]
93  pub extern "C" fn __floatundisf() {
94      loop {
95          cli();
96          halt();
97      }
98  }

```

V.XI /kernel/src/keyboard.rs

```

1  #![allow(dead_code)]
2
3  use x86::halt;
4  use pio::*;
5  use common::*;
6
7  // Keyboard ports
8  const KEYBOARD_DATA_PORT: u16 = 0x60;
9  const KEYBOARD_STATE_PORT: u16 = 0x64;
10
11  const CIRC_BUFFER_SIZE: usize = 30;
12
13  static mut BUFFER: CircBuffer = CircBuffer::new();
14  static mut SHIFT: bool = false;
15
16  struct CircBuffer {
17      buffer: [i32; CIRC_BUFFER_SIZE],
18      read: usize,
19      write: usize,
20      count: usize
21  }
22
23  pub fn keyboard_handler() {
24      unsafe {
25          let state = inb(KEYBOARD_STATE_PORT) & 1;
26          if state == 1 {
27              let mut key = inb(KEYBOARD_DATA_PORT);

```

```

28         if key >> 7 == 0 {
29             match key {
30                 LEFT_SHIFT => SHIFT = true,
31                 RIGHT_SHIFT => SHIFT = true,
32                 _ => {
33                     if SHIFT {
34                         BUFFER.write(SHIFT_KEY_MAP[key as usize] as
i32);
35                     } else {
36                         BUFFER.write(KEY_MAP[key as usize] as i32);
37                     }
38                 }
39             }
40         } else {
41             key &= !(1<<7);
42             if key == LEFT_SHIFT || key == RIGHT_SHIFT {
43                 SHIFT = false;
44             }
45         }
46     }
47 }
48
49
50 pub fn getc() -> char {
51     unsafe {
52         let mut data = BUFFER.read();
53         while data == -1 {
54             halt();
55             data = BUFFER.read();
56         }
57         return data as u8 as char;
58     }
59 }
60
61 // Non-blocking call. Return 1 if a key is pressed
62 pub fn keypressed() -> bool {
63     unsafe {
64         BUFFER.count > 0
65     }
66 }
67
68 impl CircBuffer {
69     const fn new() -> CircBuffer {
70         CircBuffer {
71             buffer: [0;CIRC_BUFFER_SIZE],
72             read: 0,
73             write: 0,
74             count: 0

```

```

75     }
76 }
77
78 fn write(&mut self, data: i32) {
79     self.buffer[self.write] = data;
80     self.write = (self.write + 1) % CIRC_BUFFER_SIZE;
81     if self.count < CIRC_BUFFER_SIZE {
82         self.count += 1;
83     }
84 }
85
86 fn read(&mut self) -> i32 {
87     if self.count > 0 {
88         let data = self.buffer[self.read];
89         self.read = (self.read + 1) % CIRC_BUFFER_SIZE;
90         self.count -= 1;
91         return data;
92     } else {
93         return -1;
94     }
95 }
96 }

```

V.XII /kernel/src/kheap.rs

```

1  #![allow(dead_code)]
2
3  use core::mem::size_of;
4  use rlibc::{memset, memcpy};
5  use paging::*;
6  use vga::*;
7
8  pub static mut KHEAP_SIZE: usize = 0x1000000;
9  pub static mut KHEAP_ADDR: u32 = 0;
10 pub static mut KHEAP_END: u32 = 0;
11
12 #[derive(Debug, Clone, Copy)]
13 #[repr(C, align(16))]
14 pub struct Header {
15     previous: u32,
16     next: u32,
17     size: usize,
18     free: bool
19 }
20
21 macro_rules! align {
22     ($size:expr) => {
23         if ($size & 0xfffff000) != $size {

```

```

24         ($size & 0xfffff000) + 0x1000
25     } else {
26         $size
27     };
28 }
29 }
30
31 pub fn kheap_init(ram_size: u32) {
32     unsafe {
33         KHEAP_ADDR = get_kernel_end();
34         KHEAP_SIZE = ((ram_size / 1000 - 1) * 0x100000 -
phys!(KHEAP_ADDR)) as usize;
35         KHEAP_END = KHEAP_ADDR + KHEAP_SIZE as u32;
36         if (KHEAP_ADDR & 0xfffff000) != KHEAP_ADDR {
37             KHEAP_ADDR &= 0xfffff000;
38             KHEAP_ADDR += 0x1000;
39         }
40         let mut entry_addr = 0;
41         INITIAL_PD.alloc_frame(&mut entry_addr, &mut phys!(KHEAP_ADDR),
KERNEL_MODE);
42         memset(entry_addr as *mut u8, 0, FRAME_SIZE);
43         memcpy(entry_addr as *mut u8, Header::null(0,
KHEAP_SIZE).as_ptr(), size_of::<Header>());
44     }
45 }
46
47 pub fn kmalloc(size: usize) -> u32 {
48     let aligned_size = align!(size) + align!(size_of::<Header>()) -
size_of::<Header>();
49     let mut addr = empty_block(aligned_size);
50     if kmalloc_table_check(addr, aligned_size) {
51         addr = empty_block(aligned_size);
52     }
53     let mut block = Header::from_ptr(addr as *mut u8);
54     if block.size >= aligned_size && block.free {
55         if block.next == 0 {
56             block.insert_tail(addr, aligned_size);
57         } else {
58             block.insert(addr, aligned_size);
59         }
60         addr += size_of::<Header>() as u32;
61         unsafe { memset(addr as *mut u8, 0, aligned_size); }
62         return addr;
63     }
64     return 0;
65 }
66
67 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] &! 0xffff) as *mut
PageTable;
135         let entry_addr = virt!((*table_ptr)[entry_idx] &! 0xffff);
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
232 u32);
233
234                     let mut tmp = 0;
235                     INITIAL_PD.alloc_frame(&mut tmp, &mut phys_addr,
236 KERNEL_MODE);
237                 }
238             }
239             self.free = false;
240             if size == self.size {
241                 memcpy(addr as *mut u8, self.as_ptr(),
242 size_of::<Header>());
243             } else {
244                 let tmp = self.next;
245                 let mut tmp_header = Header::from_ptr(tmp as *const u8);
246
247                 self.size = size;
248                 self.next = addr + total_size as u32;

```

```

245         let next_block_size = (tmp - self.next) as usize -
size_of::();
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::());
250         memcpy(self.next as *mut u8, next_header.as_ptr(),
size_of::());
251         memcpy(tmp as *mut u8, tmp_header.as_ptr(),
size_of::());
252     }
253 }
254 }
255
256 fn insert_tail(&mut self, addr: u32, size: usize) {
257     unsafe {
258         let total_size = size + size_of::();
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::();
272         let mut tail = Header::null(addr, tail_size);
273         memcpy(addr as *mut u8, self.as_ptr(), size_of::());
274         memcpy(self.next as *mut u8, tail.as_ptr(),
size_of::());
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);

```

```

284         if previous.free {
285             header_addr = self.previous;
286             self.previous = previous.previous;
287             self.size += previous.size + size_of::<Header>();
288         }
289     }
290     if self.next != 0 {
291         let mut next = Header::from_ptr(self.next as *const
u8);
292         if next.free {
293             self.next = next.next;
294             self.size += next.size + size_of::<Header>();
295         }
296         if self.next != 0 {
297             next = Header::from_ptr(self.next as *const u8);
298             if next.previous != header_addr {
299                 next.previous = header_addr;
300                 memcpy(self.next as *mut u8, next.as_ptr(),
size_of::<Header>());
301             }
302         }
303     }
304     self.free = true;
305     memcpy(header_addr as *mut u8, self.as_ptr(),
size_of::<Header>());
306     INITIAL_PD.mmap_reset_frame(header_addr / FRAME_SIZE as
u32);
307     }
308 }
309 }
310
311 fn as_ptr(&mut self) -> *const u8 {
312     self as *const Header as *const u8
313 }
314
315 fn from_ptr(ptr: *const u8) -> Header {
316     unsafe {
317         *(ptr as *const Header)
318     }
319 }
320 }

```

V.XIII /kernel/src/Makefile

```

1 ARCH = i386
2 BUILD_FOLDER = build
3
4 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 $(@)
43     $(BUILD_FOLDER)/isofiles
44     rm -r $(BUILD_FOLDER)/isofiles
45
46 $(BUILD_FOLDER)/$(KERNEL) : $(LINKER) $(OBJS) $(RUST)
47     $(LD) $(LDFLAGS) $(OBJS) $(RUST) -o $@
48
49 $(BUILD_FOLDER)/%.o : %.s
50     mkdir -p $(shell dirname $@)
51     nasm -f elf $< -o $@

```

```

52 $(RUST) :
53     $(MAKE) -C ../
54
55 $(FS) : $(SPLASH) | user
56     cargo run --manifest-path $(FS_FOLDER)/Cargo.toml $@ create MicroFS
57     1 1000000
58     cargo run --manifest-path $(FS_FOLDER)/Cargo.toml $@ add $(SPLASH)
59     cargo run --manifest-path $(FS_FOLDER)/Cargo.toml $@ add
60     $(USER_PATH)/build/hello
61     cargo run --manifest-path $(FS_FOLDER)/Cargo.toml $@ add
62     $(USER_PATH)/build/demo
63     cargo run --manifest-path $(FS_FOLDER)/Cargo.toml $@ add
64     $(USER_PATH)/build/shell
65     cargo run --manifest-path $(FS_FOLDER)/Cargo.toml $@ add
66     $(USER_PATH)/build/splash
67
68 clean :
69     rm -rf $(BUILD_FOLDER)
70
71 mrproper : clean
72     cargo clean --manifest-path $(FS_FOLDER)/Cargo.toml
73     $(MAKE) -C $(USER_PATH) clean

```

V.XIV /kernel/src/multiboot.rs

```

1  #![allow(dead_code)]
2
3  #[derive(Debug, Clone, Copy)]
4  #[repr(C)]
5  pub struct MultibootAoutSymbolTable {
6      pub tabsize: u32,
7      pub strsize: u32,
8      pub addr: u32,
9      pub reserved: u32
10 }
11
12 #[derive(Debug, Clone, Copy)]
13 #[repr(C)]
14 pub struct MultibootElfSectionHeaderTable {
15     pub num: u32,
16     pub size: u32,
17     pub addr: u32,
18     pub shndx: u32
19 }
20
21 #[derive(Debug, Clone, Copy)]
22 #[repr(C)]
23 pub struct MultibootInfo {

```

```

24  /* Multiboot info version number */
25  pub flags: u32,
26
27  /* Available memory from BIOS (in KB) */
28  pub mem_lower: u32,
29  pub mem_upper: u32,
30
31  /* "root" partition */
32  pub boot_device: u32,
33
34  /* Kernel command line */
35  pub cmdline: u32,
36
37  /* Boot-Module list */
38  pub mods_count: u32,
39  pub mods_addr: u32,
40
41  pub aout_sym: MultibootAoutSymbolTable,
42  pub elf_sec: MultibootElfSectionHeaderTable,
43
44  /* Memory Mapping buffer */
45  pub mmap_length: u32,
46  pub mmap_addr: u32,
47
48  /* Drive Info buffer */
49  pub drives_length: u32,
50  pub drives_addr: u32,
51
52  /* ROM configuration table */
53  pub config_table: u32,
54
55  /* Boot Loader Name */
56  pub boot_loader_name: u32,
57
58  /* APM table */
59  pub apm_table: u32,
60
61  /* Video */
62  pub vbe_control_info: u32,
63  pub vbe_mode_info: u32,
64  pub vbe_mode: u16,
65  pub vbe_interface_seg: u16,
66  pub vbe_interface_off: u16,
67  pub vbe_interface_len: u16
68 }
69
70 #[derive(Debug, Clone, Copy)]
71 #[repr(C)]

```

```

72 pub struct MultibootModList {
73     /* the memory used goes from bytes 'mod_start' to 'mod_end-1'
       inclusive */
74     pub mod_start: u32,
75     pub mod_end: u32,
76
77     /* Module command line */
78     pub cmdline: u32,
79
80     /* padding to take it to 16 bytes (must be zero) */
81     pub pad: u32
82 }

```

V.XV /kernel/src/paging_asm.s

```

1  %include "const.inc"
2
3  extern kernel_start
4  extern kernel_end
5  extern page_directory
6  extern kernel_pt
7
8  global load_directory
9  global get_cr3
10 global get_kernel_start
11 global get_kernel_end
12 global get_kernel_page_directory
13 global get_kernel_page_table
14
15 section .text:                ; start of the text (code) section
16
17 load_directory:
18     push ebp
19     mov ebp, esp
20
21     mov eax,[esp+8]            ; Get the pointer to the page directory, passed
as a parameter.
22     mov cr3, eax
23
24     mov ebx, cr0               ; read current cr0
25     or  ebx, 1 << 31          ; set PG
26     mov cr0, ebx               ; update cr0
27
28     leave
29     ret
30
31 get_cr3:
32     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

```
1  #![allow(dead_code)]
2  #![macro_use]
3
4  use core::mem::size_of;
5  use core::ops::{Index, IndexMut};
6  use rlibc::memset;
7  use vga::*;
8  use kheap::*;
9
10 pub const KERNEL_BASE: u32 = 0xC0000000;
11 pub const KERNEL_PAGE_NUMBER: u32 = KERNEL_BASE >> 22;
12
13 const MMAP_SIZE: usize = 0x20000;
14 const MEMORY_FSIZE: usize = 0x100000;
15 pub const TABLE_FSIZE: usize = 0x400;
16 pub const TABLE_SIZE: usize = 0x400000;
17 pub const FRAME_SIZE: usize = 0x1000;
18
19 pub const KERNEL_MODE: u32 = 0x0;
20 pub const USER_MODE: u32 = 0x4;
21
22 static mut INITIAL_MMAP: [u8;MMAP_SIZE] = [0;MMAP_SIZE];
23 pub static mut INITIAL_PD: PageDirectory = PageDirectory::null();
24 pub static mut USER_PD: *mut PageDirectory = 0 as *mut PageDirectory;
25
26 #[derive(Clone, Copy)]
27 #[repr(C, align(4096))]
28 pub struct PageDirectory {
29     pub tables: *mut PageTable,
30     pub mmap: *mut [u8;MMAP_SIZE]
31 }
32
33 #[derive(Clone, Copy)]
34 #[repr(C, align(4096))]
35 pub struct PageTable {
36     pub entries: [u32;TABLE_FSIZE]
37 }
38
39 extern "C" {
40     pub fn load_directory(pd_addr: u32);
41     pub fn get_cr3() -> u32;
42     pub fn get_kernel_start() -> u32;
43     pub fn get_kernel_end() -> u32;
44     fn get_kernel_page_directory() -> u32;
45     fn get_kernel_page_table() -> u32;
46 }
```

```

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],
200     [u32;MMAP_SIZE/4]>(mmap) };
201     let size = TABLE_FSIZE / 8;
202     let mmap_idx = size * idx;
203     let mut is_empty = true;
204
205     for i in mmap_idx..(mmap_idx+size) {
206         if unsafe { (*self.mmap)[i] } != 0 {
207             is_empty = false;
208             break;
209         }
210     }
211     self.mmap_set_frame(table_idx);
212     return is_empty;
213 }
214
215 pub fn mmap_alloc_frame(&mut self, addr: u32) -> u32 {
216     let frame = if addr == 0 {
217         self.mmap_get_free_frame()
218     } else {
219         addr / FRAME_SIZE as u32
220     };
221     self.mmap_set_frame(frame);
222     return frame * FRAME_SIZE as u32;
223 }
224
225 pub fn mmap_get_free_frame(&mut self) -> u32 {
226     for i in 0..MEMORY_FSIZE {
227         if self.mmap_frame_state(i as u32) == 0 {
228             return i as u32;
229         }
230     }
231     return 0;
232 }

```

```

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
}

```

```

279     }
280 }
281
282 impl Index<usize> for PageTable {
283     type Output = u32;
284
285     fn index(&self, index: usize) -> &u32 {
286         &self.entries[index]
287     }
288 }
289
290 impl IndexMut<usize> for PageTable {
291     fn index_mut(&mut self, index: usize) -> &mut u32 {
292         &mut self.entries[index]
293     }
294 }
295
296 impl PageTable {
297     fn null() -> PageTable {
298         PageTable {
299             entries: [0;TABLE_FSIZE]
300         }
301     }
302
303     pub fn from_ptr(addr: u32) -> *mut PageTable {
304         addr as *mut PageTable
305     }
306
307     pub fn as_ptr(&mut self) -> u32 {
308         self as *const PageTable as u32
309     }
310 }

```

V.XVII /kernel/src/pic.rs

```

1  #![allow(dead_code)]
2
3  use pio::*;
4
5  // PIC ports
6  const PIC1_CMD: u16 = 0x20;
7  const PIC1_DATA: u16 = 0x21;
8  const PIC2_CMD: u16 = 0xA0;
9  const PIC2_DATA: u16 = 0xA1;
10
11 // End Of Interrupt (reactivate the specified PIC)
12 const PIC_EOI: u8 = 0x20;
13

```

```

14 // Initialize the PICs by remapping IRQs 0-15 to 32-47
15 // More details here: http://wiki.osdev.org/8259\_PIC
16 pub fn pic_init() {
17     // By default IRQ 0 to 7 (master PIC) are mapped to interrupts 0-7
18     // and IRQ 8 to 15 (slave PIC) are mapped to interrupts 8-15.
19     // In protected mode, this scheme conflicts with CPU exceptions
    wich are
20     // reserved by the CPU and mapped to interrupts 0 to 31.
21     // Therefore, we remap IRQ 0-7 to interrupts 32-39 and
22     // IRQ 8-15 to interrupts 40-47
23
24     unsafe {
25         // Restart both PICs
26         outb(PIC1_CMD, 0x11);
27         outb(PIC2_CMD, 0x11);
28
29         // Remap IRQ [0..7] to [32..39]
30         outb(PIC1_DATA, 32);
31
32         // Remap IRQ [8..15] to [40..47]
33         outb(PIC2_DATA, 40);
34
35         // Setup cascading
36         outb(PIC1_DATA, 0x04);
37         outb(PIC2_DATA, 0x02);
38         outb(PIC1_DATA, 0x01);
39         outb(PIC2_DATA, 0x01);
40     }
41 }
42
43 // Send an end-of-interrupt to the PICs.
44 // An EOI must also be sent to the slave for IRQs > 7
45 pub fn pic_eoi(irq: u32) {
46     unsafe {
47         if irq > 7 {
48             outb(PIC2_CMD, PIC_EOI);
49         }
50         outb(PIC1_CMD, PIC_EOI);
51     }
52 }

```

V.XVIII /kernel/src/pio_asm.s

```

1 global outb
2 global outw
3 global inb
4 global inw
5

```



```

6  section .txt
7
8  outb:
9      push ebp
10     mov ebp, esp
11
12     mov word dx, [esp+8]
13     mov byte al, [esp+12]
14     out dx, al
15
16     mov eax, 0
17     leave
18     ret
19
20 outw:
21     push ebp
22     mov ebp, esp
23
24     mov word dx, [esp+8]
25     mov word ax, [esp+12]
26     out dx, ax
27
28     mov eax, 0
29     leave
30     ret
31
32 inb:
33     push ebp
34     mov ebp, esp
35
36     mov word dx, [esp+8]
37     in byte al, dx
38
39     leave
40     ret
41
42 inw:
43     push ebp
44     mov ebp, esp
45
46     mov word dx, [esp+8]
47     in word ax, dx
48
49     leave
50     ret

```

V.XIX /kernel/src/pio.rs

```
1  #![allow(dead_code)]
2
3  // CRTC ports
4  const CRTC_CMD: u16 = 0x3d4;
5  const CRTC_DATA: u16 = 0x3d5;
6  // CRTC registers
7  const CRTC_START_REG: u8 = 0xa;
8  const CRTC_LOCATION_MSB: u8 = 0xe;
9  const CRTC_LOCATION_LSB: u8 = 0xf;
10
11  extern "C" {
12      pub fn outb(port: u16, data: u8);
13      pub fn outw(port: u16, data: u16);
14      pub fn inb(port: u16) -> u8;
15      pub fn inw(port: u16) -> u16;
16  }
17
18  pub fn move_cursor(position: u16) {
19      unsafe {
20          let pos_msb = ((position >> 8) & 0xff) as u8;
21          let pos_lsb = (position & 0xff) as u8;
22          outb(CRTC_CMD, CRTC_LOCATION_MSB);
23          outb(CRTC_DATA, pos_msb);
24          outb(CRTC_CMD, CRTC_LOCATION_LSB);
25          outb(CRTC_DATA, pos_lsb);
26      }
27  }
28
29  pub fn enable_cursor() {
30      unsafe {
31          outb(CRTC_CMD, CRTC_START_REG);
32          let reg = inb(CRTC_DATA);
33          outb(CRTC_CMD, CRTC_START_REG);
34          outb(CRTC_DATA, reg & !0x20);
35      }
36  }
37
38  pub fn disable_cursor() {
39      unsafe {
40          outb(CRTC_CMD, CRTC_START_REG);
41          let reg = inb(CRTC_DATA);
42          outb(CRTC_CMD, CRTC_START_REG);
43          outb(CRTC_DATA, reg | 0x20);
44      }
45  }
```

V.XX /kernel/src/syscall_asm.s

```
1  %include "const.inc"
2
3  section .text                ; start of the text (code) section
4  align 4                     ; the code must be 4 byte aligned
5
6  extern syscall_handler
7
8  global _syscall_handler
9
10 _syscall_handler:
11     ; Save all registers
12     push    ebx
13     push    ecx
14     push    edx
15     push    esi
16     push    edi
17     push    ebp
18     push    ds
19     push    es
20     push    fs
21     push    gs
22
23     ; Load kernel data descriptor into all segments
24     push    eax
25     mov     ax,GDT_KERNEL_DATA_SELECTOR
26     mov     ds,ax
27     mov     es,ax
28     mov     fs,ax
29     mov     gs,ax
30     pop     eax
31
32     ; Pass the 6 arguments (nb, arg1, etc.) to the syscall_handler
33     ; They are in reverse order to match gcc's IA-32 ABI.
34
35     ; use edi (last arg) to store the task's TSS selector the syscall
36     ; originated
37     ; from, so the kernel can properly address the memory associated to
38     ; the calling task
39     ; (by applying the right offset to addresses passed in arguments)
40     str     edi ; store the segment selector from the task register (TR)
41     in edi
42     push    edi
43
44     push    esi
45     push    edx
46     push    ecx
```

```

44     push    ebx
45     push    eax
46
47     call    syscall_handler
48
49     ; These 6 "pop eax" instructions are only here to balance the pushes
50     ; above used to pass the arguments to the syscall_handler function
51     pop     ebx
52     pop     ebx
53     pop     ebx
54     pop     ebx
55     pop     ebx
56     pop     ebx
57
58     ; Restore all registers
59     pop     gs
60     pop     fs
61     pop     es
62     pop     ds
63     pop     ebp
64     pop     edi
65     pop     esi
66     pop     edx
67     pop     ecx
68     pop     ebx
69     iret

```

V.XXI /kernel/src/syscall.rs

```

1  #![allow(dead_code)]
2
3  use vga::*;
4  use pio::*;
5  use timer::*;
6  use keyboard::*;
7  use fs::*;
8  use task::*;
9  use paging::FRAME_SIZE;
10 use kheap::*;
11 use common::*;
12
13 extern "C" {
14     pub fn _syscall_handler();
15 }
16
17 /// System call handler: call the appropriate system call according to
the nb argument.
18 /// Called by the assembly code _syscall_handler

```

```

19  #[no_mangle]
20  pub unsafe extern fn syscall_handler(nb: Syscall, _arg1: u32, _arg2:
u32, _arg3: u32, _arg4: u32) -> i32 {
21      let addr = 0;
22      match nb {
23          Syscall::Puts => syscall_puts(addr, _arg1),
24          Syscall::Putc => syscall_putc(_arg1),
25          Syscall::Exec => syscall_exec(addr, _arg1),
26          Syscall::Keypressed => syscall_keypressed(),
27          Syscall::Getc => syscall_getc(),
28          Syscall::FileStat => syscall_file_stat(addr, _arg1, addr + _arg2),
29          Syscall::FileOpen => syscall_file_open(addr, _arg1),
30          Syscall::FileClose => syscall_file_close(_arg1),
31          Syscall::FileRead => syscall_file_read(_arg1, addr + _arg2,
_arg3),
32          Syscall::FileSeek => syscall_file_seek(_arg1, _arg2),
33          Syscall::FileIterator => syscall_file_iterator(addr + _arg1),
34          Syscall::FileNext => syscall_file_next(addr, _arg1, addr + _arg2),
35          Syscall::GetTicks => syscall_get_ticks(),
36          Syscall::Sleep => syscall_sleep(_arg1),
37          Syscall::SetCursor => syscall_set_cursor(_arg1, _arg2),
38          Syscall::GetCursor => syscall_get_cursor(addr + _arg1, addr +
_arg2),
39          Syscall::CursorDisable => syscall_cursor_disable(_arg1),
40          Syscall::CopyScr => syscall_copy_scr(addr + _arg1),
41          Syscall::AllocFrame => syscall_alloc_frame(),
42          Syscall::FreeFrame => syscall_free_frame(_arg1),
43      }
44  }
45
46  unsafe fn syscall_puts(base_addr: u32, string_offset: u32) -> i32 {
47      let mut string = *((base_addr + string_offset) as *mut String);
48      string.offset(base_addr);
49      vga_write_str(string.to_string());
50      return 0;
51  }
52
53  unsafe fn syscall_putc(c: u32) -> i32 {
54      vga_write_byte(c as u8);
55      return 0;
56  }
57
58  unsafe fn syscall_exec(base_addr: u32, string_offset: u32) -> i32 {
59      let mut string = *((base_addr + string_offset) as *mut String);
60      string.offset(base_addr);
61      exec(string.to_string()) as i32
62  }
63

```

```

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,
73 stat_addr: u32) -> i32 {
74     let mut string = *((base_addr + string_offset) as *mut String);
75     string.offset(base_addr);
76     let stat = stat_addr as *mut Stat;
77     *stat = Stat::new(string.to_string());
78     if (*stat).start == 0 {
79         return -1;
80     }
81     return 0;
82 }
83
84 unsafe fn syscall_file_open(base_addr: u32, string_offset: u32) -> i32 {
85     let mut string = *((base_addr + string_offset) as *mut String);
86     string.offset(base_addr);
87     file_open(string.to_string())
88 }
89
90 unsafe fn syscall_file_close(fd: u32) -> i32 {
91     file_close(fd as i32)
92 }
93
94 unsafe fn syscall_file_read(fd: u32, buf_addr: u32, n: u32) -> i32 {
95     file_read(fd as i32, buf_addr as *mut u8, n as usize)
96 }
97
98 unsafe fn syscall_file_seek(fd: u32, offset: u32) -> i32 {
99     file_seek(fd as i32, offset as usize)
100 }
101
102 unsafe fn syscall_file_iterator(it_addr: u32) -> i32 {
103     let it = it_addr as *mut FileIterator;
104     *it = FileIterator::new();
105     return 0;
106 }
107
108 unsafe fn syscall_file_next(base_addr: u32, string_offset: u32, it_addr:
109 u32) -> i32 {
110     let bytes = (base_addr + string_offset) as *mut u8;
111     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

```
1 global task_ltr
2 global task_switch
3
4 section .data
5 tss_sel_offs dd 0 ; must always be 0
6 tss_sel_seg dw 0 ; overwritten by the task_switch function
7
8 section .text: ; start of the text (code) section
9 align 4 ; the code must be 4 byte aligned
10
11 ; Load the task register with the TSS selector passed in argument.
12 ; The TSS selector represents the offset of the TSS descriptor in the GDT
13 ; table.
14 ; void load_task_register(uint16_t tss_selector);
15 ; NOTE: The GDT must be loaded before issuing the ltr instruction!
16 ;
17 ; void task_ltr(uint16_t tss_selector);
18 task_ltr:
19     mov     eax,[esp+4]
20     ltr     ax
21     ret
22
23 ; Call the task specified by the tss selector in argument.
24 ; When the CPU switches to the new task, it automatically loads the task
25 ; register
26 ; with the new task (ltr instruction) and the LDT from the
27 ; tss.ldt_selector field.
28 ;
29 ; void task_switch(uint16_t tss_selector)
30 task_switch:
31     mov     ax,[esp+4] ; get the TSS selector passed in argument (16
32 ; bits)
33     ; rewrite the segment to jump to with the tss selector passed in
34 ; argument
35     mov     ecx,tss_sel_seg
36     mov     [ecx],ax
37     call    far [ecx-4]
38     ret
```

V.XXIII /kernel/src/task.rs

```
1 #![allow(dead_code)]
2
3 use core::mem::size_of;
4 use x86::*;
5 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] =
17 [0;STACK_SIZE];
18 pub static mut TASKS: [Task;TASKS_NB] = [Task::new();TASKS_NB];
19
20 #[derive(Clone, Copy)]
21 #[repr(C)]
22 pub struct Task {
23     pub tss: Tss,
24     pub tss_selector: u16,
25     pub kernel_stack: [u8;STACK_SIZE],
26     pub is_free: bool,
27     pub pd: PageDirectory
28 }
29
30 #[derive(Debug, Clone, Copy)]
31 #[repr(C, packed)]
32 pub struct Tss {
33     previous_task_link: u16, reserved0: u16,
34     esp0: u32,
35     ss0: u16, reserved1: u16,
36     esp1: u32,
37     ss1: u16, reserved2: u16,
38     esp2: u32,
39     ss2: u16, reserved3: u16,
40     cr3: u32,
41     eip: u32, eflags: u32, eax: u32, ecx: u32, edx: u32,
42     ebx: u32, esp: u32, ebp: u32, esi: u32, edi: u32,
43     es: u16, reserved4: u16,
44     cs: u16, reserved5: u16,
45     ss: u16, reserved6: u16,
46     ds: u16, reserved7: u16,
47     fs: u16, reserved8: u16,
48     gs: u16, reserved9: u16,
49     ldt_selector: u16, reserved10: u16,
50     reserved11: u16,
51     iomap_base_addr: u16 // adresse (relative to byte 0 of the TSS) of
the IO permission bitmap
}

```

```

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
103         u32;
104         TASKS[idx as usize].tss.ebp = stack_addr + STACK_SIZE as
105         u32;
106         TASKS[idx as usize].tss.cr3 = phys!(TASKS[idx as
107         usize].pd.tables as u32);
108
109         task_switch(TASKS[idx as usize].tss_selector as u16);
110         switch_directory(&mut INITIAL_PD);
111         // re-load original directory and free memory
112         TASKS[idx as usize].is_free = true;
113         ufree(code_addr);
114         ufree(stack_addr);
115         switch_directory(pd_backup);
116         TASKS[idx as usize].pd.free();
117         return 0;
118     } else {
119         println!("exec: {}: not found", filename);
120     }
121 }
122 } else {
123     println!("exec: no free task slot found");
124 }
125 return -1;
126 }
127 }
128
129 fn free_task() -> i8 {
130     unsafe {
131         let mut cnt = 0;
132         for task in TASKS.iter() {
133             if task.is_free {
134                 return cnt;
135             }
136             cnt += 1;
137         }
138         return -1;
139     }
140 }
141
142 impl Task {
143     const fn new() -> Task {
144         Task {
145             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
151     usize)) / size_of::<Task>();
152     // Add the task's TSS to the GDT
153     let tss = &self.tss as *const _ as u32;
154     GDT[GDT_SIZE + idx] = GdtEntry::make_tss(tss, DPL_KERNEL);
155
156     // Initialize the TSS fields
157     self.tss_selector = GDT[GDT_SIZE + idx].to_selector() as u16;
158
159     // Code and data segment selectors are in the LDT
160     let cs = (GDT_USER_CODE_SELECTOR | DPL_USER) as u32;
161     let ds = (GDT_USER_DATA_SELECTOR | DPL_USER) as u32;
162     self.tss.cs = cs as u16;
163     self.tss.ds = ds as u16;
164     self.tss.es = self.tss.ds;
165     self.tss.fs = self.tss.ds;
166     self.tss.gs = self.tss.ds;
167     self.tss.ss = self.tss.ds;
168     self.tss.eflags = 512; // Activate hardware interrupts (bit 9)
169
170     // Task's kernel stack
171     self.tss.ss0 = GDT_KERNEL_DATA_SELECTOR as u16;
172     self.tss.esp0 = (&self.kernel_stack as *const _ as usize +
173     STACK_SIZE) as u32;
174 }
175
176 impl Tss {
177     const fn new() -> Tss {
178         Tss {
179             previous_task_link: 0, reserved0: 0,
180             esp0: 0,
181             ss0: 0, reserved1: 0,
182             esp1: 0,
183             ss1: 0, reserved2: 0,
184             esp2: 0,
185             ss2: 0, reserved3: 0,
186             cr3: 0,
187             eip: 0, eflags: 0, eax: 0, ecx: 0, edx: 0,
188             ebx: 0, esp: 0, ebp: 0, esi: 0, edi: 0,

```

```

188         es: 0, reserved4: 0,
189         cs: 0, reserved5: 0,
190         ss: 0, reserved6: 0,
191         ds: 0, reserved7: 0,
192         fs: 0, reserved8: 0,
193         gs: 0, reserved9: 0,
194         ldt_selector: 0, reserved10: 0,
195         reserved11: 0,
196         iomap_base_addr: 0
197     }
198 }
199 }

```

V.XXIV /kernel/src/timer.rs

```

1  #![allow(dead_code)]
2
3  use x86::halt;
4  use core::u32;
5  use pio::outb;
6
7  // PIT ports
8  const PIT_CMD: u16 = 0x43;
9  const PIT_CANAL_0: u16 = 0x40;
10 const PIT_DIV_REPEAT: u8 = 0x36;
11
12 pub const MIN_FREQ: u32 = 19;
13 pub const MAX_FREQ: u32 = 1193180;
14
15 static mut TIMER: Timer = Timer { freq: 0, ticks: 0 };
16
17 pub fn timer_init(freq_hz: u32) {
18     unsafe { TIMER.init(freq_hz) }
19 }
20
21 pub fn timer_handler() {
22     unsafe { TIMER.ticks+=1 }
23 }
24
25 pub fn get_freq() -> u32 {
26     unsafe { return TIMER.freq }
27 }
28
29 pub fn get_ticks() -> u32{
30     unsafe { return TIMER.ticks }
31 }
32
33 pub fn sleep(ms: u32) {

```

```

34     let duration = get_ticks() + (ms * unsafe { TIMER.freq } / 1000);
35     loop {
36         if get_ticks() >= duration {
37             break;
38         }
39         halt();
40     }
41 }
42
43 struct Timer {
44     freq: u32,
45     ticks: u32
46 }
47 impl Timer {
48     pub fn init(&mut self, freq_hz: u32) {
49         match freq_hz {
50             0...MIN_FREQ => self.freq = MIN_FREQ,
51             MAX_FREQ...u32::MAX => self.freq = MAX_FREQ,
52             _ => self.freq = freq_hz
53         }
54
55         #[cfg(not(test))]
56         unsafe {
57             let div = MAX_FREQ / self.freq;
58             // divisor selection and repetition mode
59             outb(PIT_CMD, PIT_DIV_REPEAT);
60             // divisor LSB on canal 0
61             outb(PIT_CANAL_0, (div & 0xFF) as u8);
62             // divisor MSB on canal 0
63             outb(PIT_CANAL_0, (div >> 8) as u8);
64         }
65     }
66
67     pub fn get_freq(&mut self) -> u32 {
68         return self.freq;
69     }
70
71     pub fn get_ticks(&mut self) -> u32 {
72         return self.ticks;
73     }
74 }

```

V.XXV /kernel/src/vga.rs

```

1  #![allow(dead_code)]
2  #![macro_use]
3
4  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);

```



```

100         }
101     }
102     self.set_cursor(0, 0);
103 }
104 }
105
106 fn write_byte(&mut self, byte: u8) {
107     if byte == b'\n' || self.cursor_x >= BUFFER_WIDTH {
108         if self.cursor_y == BUFFER_HEIGHT-1 {
109             self.shift_up();
110             self.cursor_x = 0;
111         } else {
112             self.cursor_x = 0;
113             self.cursor_y += 1;
114         }
115     }
116     match byte {
117         b'\0' => return,
118         b'\n' => (),
119         b'\t' => {
120             for _i in 0..TAB_SIZE {
121                 self.write_byte(b' ');
122             }
123         }
124         0x8 => {
125             if self.cursor_x > 0 {
126                 self.cursor_x -= 1;
127             } else if self.cursor_y > 0 {
128                 self.cursor_y -= 1;
129                 self.cursor_x = BUFFER_WIDTH-1;
130             }
131             unsafe { (*self.buffer)[self.cursor_y][self.cursor_x] =
Character::new(b'\0', self.attribute); }
132         }
133         _ => {
134             unsafe { (*self.buffer)[self.cursor_y][self.cursor_x] =
Character::new(byte, self.attribute); }
135             self.cursor_x += 1;
136         }
137     }
138     move_cursor(self.get_pos());
139 }
140
141 fn write_str(&mut self, buf: &str) {
142     for byte in buf.bytes() {
143         self.write_byte(byte);
144     }
145 }

```

```

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;

```

```

13 pub const TYPE_CODE_EXECONLY: u8 = 9;
14 pub const TYPE_CODE_EXECREAD: u8 = 11;
15
16 // Descriptor types for system segments and gates
17 pub const TYPE_LDT: u8 = 2;
18 pub const TYPE_TASK_GATE: u8 = 5;
19 pub const TYPE_TSS: u8 = 9;
20 pub const TYPE_CALL_GATE: u8 = 12;
21 pub const TYPE_TRAP_GATE: u8 = 15;
22 pub const TYPE_INTERRUPT_GATE: u8 = 14;
23
24 // Descriptor system bit (S)
25 // For code or data segments
26 pub const S_CODE_OR_DATA: u8 = 1;
27 // For TSS segment, LDT, call gate, interrupt gate, trap gate, task gate
28 pub const S_SYSTEM: u8 = 0;
29
30 // D/B bit
31 pub const DB_SEG: u8 = 1;
32 pub const DB_SYS: u8 = 0;
33
34 // kernel code and data selectors in the GDT
35 pub const GDT_KERNEL_CODE_SELECTOR: u8 = 0x08;
36 pub const GDT_KERNEL_DATA_SELECTOR: u8 = 0x10;
37 pub const GDT_USER_CODE_SELECTOR: u8 = 0x18;
38 pub const GDT_USER_DATA_SELECTOR: u8 = 0x20;
39
40 // Disable hardware interrupts.
41 pub fn cli() {
42     unsafe { asm!("cli"); }
43 }
44
45 // Enable hardware interrupts.
46 pub fn sti() {
47     unsafe { asm!("sti"); }
48 }
49
50 // Halt the processor.
51 // External interrupts wake up the CPU, hence the cli instruction.
52 pub fn halt() {
53     unsafe { asm!("hlt"); }
54 }

```

V.XXVII /kernel/Cargo.toml

```

1 [package]
2 name = "rust_os"
3 version = "0.1.0"

```

```

4 authors = ["orpheeantoniadis <orphee.antoniadis@gmail.com>"]
5
6 [lib]
7 name = "kernel"
8 path = "src/kernel.rs"
9 crate-type = ["staticlib"]
10
11 [dependencies]
12 rlibc = "1.0"
13 common = { path = "../common" }
14
15 [profile.release]
16 lto = true
17 panic = 'abort'

```

V.XXVIII /kernel/i386-rust_os.json

```

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

```

V.XXIX /kernel/Makefile

```

1 ARCH = i386
2 CC = xargo
3 TARGET = $(ARCH)-rust_os
4 FLAGS = -v --target $(TARGET)
5 SOURCE_PATH = src/
6
7 .PHONY: all build run test clean mrproper
8
9 all: build
10
11 build:
12     RUST_TARGET_PATH=$(shell pwd) $(CC) build $(FLAGS)
13     $(MAKE) -C $(SOURCE_PATH)
14

```

```
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/MicroFS/src/micro_fs/add.rs

```
1 use super::*;
2
3 impl MicroFS {
4     pub fn add(&mut self, path: &str) {
5         // read file
6         let mut file_buffer = Vec::new();
7         let mut file = match File::open(path) {
8             Ok(f) => f,
9             Err(_) => {
10                 println!("File not found.");
11                 return;
12             }
13         };
14         file.read_to_end(&mut file_buffer).expect("Something went wrong
when reading the file !");
15
16         let mut entry = Entry::new(path);
17         entry.size = file_buffer.len() as u32;
18         let entry_block_size = file_buffer.len() / (SECTOR_SIZE *
self.sb.block_size as usize) + 1;
19         println!("Entry size (bytes) = {}", file_buffer.len());
20         println!("Entry size (sectors) = {}", file_buffer.len() /
SECTOR_SIZE + 1);
21         println!("Entry size (blocks) = {}", entry_block_size);
22
23         let mut blocks = self.empty_blocks(&mut entry);
24         if blocks.len() < entry_block_size {
25             println!("\nFile too large for File System.");
26         } else {
27             self.new_entries.push(entry);
28             self.update_fat(&mut blocks, true);
29         }
30     }
31 }
```

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

```
1 use super::*;
2
3 impl MicroFS {
4     pub fn create(&mut self, label: &str, bs: u8, size: usize) {
5         let mut file = File::create(self.image.clone()).expect("Failed to
create file!");
6         self.sb = SuperBlock::new(label, bs, size);
7     }
8 }
```

```

7      self.write_super_block(&mut file);
8      println!("Super block written to image.");
9      for _i in 0..self.sb.fat_size {
10         file.write_all(&[0xff;1]).expect("Failed to write in file!");
11     }
12     self.set_fat();
13     println!("FAT written to image.");
14     file.seek(SeekFrom::Start(self.root_entry() as u64)).expect("File
seek failed !");
15     let rest = (size / SECTOR_SIZE) - (self.sb.root_entry as usize *
self.sb.block_size as usize);
16     for _i in 0..rest {
17         file.write_all(&[0;SECTOR_SIZE]).expect("Failed to write in
file!");
18     }
19     // self.set_entries();
20     println!("Rest of image written.");
21     println!("Total bytes = {}", (size / SECTOR_SIZE) * SECTOR_SIZE);
22     println!("Total sectors = {}", size / SECTOR_SIZE);
23     println!("Total blocks = {}", size / (SECTOR_SIZE * bs as usize));
24 }

25
26 fn write_super_block(&mut self, file: &mut File) {
27     unsafe {
28         file.write_all(&[0;11]).expect("Failed to write in file!");
29         file.write_all(&(mem::transmute:::<u16, [u8;
2]>(self.sb.sector_size))).expect("Failed to write in file!");
30         file.write_all(&(mem::transmute:::<u8, [u8;
1]>(self.sb.block_size))).expect("Failed to write in file!");
31         file.write_all(&[0;22]).expect("Failed to write in file!");
32         file.write_all(&(mem::transmute:::<u32, [u8;
4]>(self.sb.fat_size))).expect("Failed to write in file!");
33         file.write_all(&[0;2]).expect("Failed to write in file!");
34         file.write_all(&(mem::transmute:::<u16, [u8;
2]>(self.sb.version))).expect("Failed to write in file!");
35         file.write_all(&(mem::transmute:::<u32, [u8;
4]>(self.sb.root_entry))).expect("Failed to write in file!");
36         file.write_all(&[0;34]).expect("Failed to write in file!");
37         file.write_all(&(self.sb.label)).expect("Failed to write in
file!");
38         file.write_all(&[0;420]).expect("Failed to write in file!");
39         file.write_all(&(mem::transmute:::<u16, [u8;
2]>(self.sb.signature))).expect("Failed to write in file!");
40     }
41 }
42 }

```

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

```
1 use super::*;
2
3 impl MicroFS {
4     pub fn del(&mut self, filename: &str) {
5         for entry in self.entries.clone() {
6             if bytes_to_str(&entry.name) == filename.clone() {
7                 let mut blocks = self.get_blocks(&mut entry.clone());
8                 self.update_fat(&mut blocks, false);
9             }
10        }
11        self.entries.retain(|e| bytes_to_str(&e.name) !=
filename.clone());
12        for entry in self.new_entries.clone() {
13            if bytes_to_str(&entry.name) == filename.clone() {
14                let mut blocks = self.get_blocks(&mut entry.clone());
15                self.update_fat(&mut blocks, false);
16            }
17        }
18        self.new_entries.retain(|e| bytes_to_str(&e.name) !=
filename.clone());
19    }
20 }
```

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

```
1 use super::*;
2
3 impl MicroFS {
4     pub fn info(&mut self) {
5         println!("\nFS info :");
6         println!("Label : {}", bytes_to_str(&self.sb.label));
7         println!("Version : {}", self.sb.version);
8         println!("Sector size : {} bytes", self.sb.sector_size);
9         println!("Block size : {} sector(s)", self.sb.block_size);
10        println!("FAT size : {} bytes", self.sb.fat_size);
11        println!("Root entry : {}", self.sb.root_entry);
12    }
13 }
```

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

```
1 use super::*;
2
3 impl MicroFS {
4     pub fn list(&mut self) {
```



```

5     println!("\nFiles :");
6     for entry in self.entries.clone() {
7         println!("{}", bytes_to_str(&entry.name));
8         println!("{}", bytes_to_str(&entry.size));
9     }
10    for entry in self.new_entries.clone() {
11        println!("{}", bytes_to_str(&entry.name));
12        println!("{}", bytes_to_str(&entry.size));
13    }
14 }
15 }

```

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

```

1 use std::mem;
2 use std::str;
3 use std::io::prelude::*;
4 use std::io::SeekFrom;
5 use std::fs;
6 use std::fs::File;
7 use std::fs::OpenOptions;
8
9 pub mod utils;
10 use self::utils::*;
11
12 mod create;
13 mod add;
14 mod save;
15 mod del;
16 mod list;
17 mod info;
18
19 pub const MAGIC: u16 = 0x55aa;
20 pub const SECTOR_SIZE: usize = 0x200;
21
22 #[derive(Debug)]
23 pub struct MicroFS {
24     pub image: String,
25     pub sb: SuperBlock,
26     pub fat: Vec<u8>,
27     pub entries: Vec<Entry>,
28     pub new_entries: Vec<Entry>
29 }
30 impl MicroFS {
31     pub fn new(image: &str) -> MicroFS {
32         unsafe {
33             if fs::metadata(image).is_ok() {
34                 let mut raw_sb = [0; SECTOR_SIZE];

```

```

35         let mut file = File::open(image).expect("File not found
!");
36         file.read(&mut raw_sb).expect("Something went wrong
reading the file !");
37         let signature = mem::transmute::<[u8;2],
u16>([raw_sb[510], raw_sb[511]]);
38         if signature == MAGIC {
39             // get super block
40             let bs = raw_sb[13];
41             let raw_label: Vec<u8> = Vec::from(&(raw_sb[82..90]));
42             let label = String::from_utf8(raw_label).unwrap();
43             let size = fs::metadata(image.clone()).expect("Failed
getting metadata!").len() as usize;
44             let sb = SuperBlock::new(&label, bs, size);
45             let mut fs = MicroFS {
46                 image: image.to_string(),
47                 sb: sb,
48                 fat: Vec::new(),
49                 entries: Vec::new(),
50                 new_entries: Vec::new()
51             };
52             fs.set_fat();
53             fs.set_entries();
54             println!("\n{} is a valid image. You can modify it
using the menu.", image);
55             return fs;
56         }
57     }
58     println!("\n{} does not exist. You should create it first
with the menu.", image);
59     return MicroFS {
60         image: image.to_string(),
61         sb: mem::uninitialized(),
62         fat: Vec::new(),
63         entries: Vec::new(),
64         new_entries: Vec::new()
65     };
66 }
67 }
68 }
69
70 #[derive(Debug, Copy, Clone)]
71 #[repr(C)]
72 pub struct SuperBlock {
73     pub sector_size: u16,
74     pub block_size: u8,
75     pub fat_size: u32,
76     pub version: u16,

```

```

77     pub root_entry: u32,
78     pub label: [u8;8],
79     pub signature: u16
80 }
81 impl SuperBlock {
82     pub fn new(label: &str, bs: u8, size: usize) -> SuperBlock {
83         let mut raw_label : [u8;8] = [0;8];
84         let mut i = 0;
85         for byte in label.bytes() {
86             raw_label[i] = byte;
87             i += 1;
88             if i == 8 { break; }
89         }
90         let fat_size = size / (SECTOR_SIZE * bs as usize);
91         let mut root_entry = (SECTOR_SIZE + fat_size) / (SECTOR_SIZE * bs
as usize);
92         if ((SECTOR_SIZE + fat_size) % (SECTOR_SIZE * bs as usize)) != 0 {
93             root_entry += 1;
94         }
95         SuperBlock {
96             sector_size: SECTOR_SIZE as u16,
97             block_size: bs,
98             fat_size: fat_size as u32,
99             version: 1,
100            root_entry: root_entry as u32,
101            label: raw_label,
102            signature: MAGIC
103        }
104     }
105 }
106
107 #[derive(Debug, Copy, Clone)]
108 #[repr(C)]
109 pub struct Entry {
110     pub name: [u8;26],
111     pub start: u16,
112     pub size: u32
113 }
114 impl Entry {
115     fn new(name: &str) -> Entry {
116         let mut raw_name : [u8;26] = [0;26];
117         let mut i = 0;
118         for byte in name.bytes() {
119             raw_name[i] = byte;
120             i += 1;
121             if i == 26 { break; }
122         }
123         Entry {

```

```

124         name: raw_name,
125         start: 0,
126         size: 0
127     }
128 }
129 }

```

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

```

1  use std::path::Path;
2  use super::*;
3
4  impl MicroFS {
5      pub fn save(&mut self) {
6          unsafe {
7              let mut image =
OpenOptions::new().read(true).write(true).open(self.image.clone()).expect("File
not found !");
8              image.seek(SeekFrom::Start(SECTOR_SIZE as u64)).expect("File
seek failed !");
9              image.write_all(&(self.fat)).expect("Failed to write in
file!");
10
11              image.seek(SeekFrom::Start(self.root_entry() as
u64)).expect("File seek failed !");
12              for _i in 0..(self.entries_size() / SECTOR_SIZE) {
13                  image.write_all(&[0;SECTOR_SIZE]).expect("Failed to write
in file!");
14              }
15
16              image.seek(SeekFrom::Start(self.root_entry() as
u64)).expect("File seek failed !");
17              for entry in self.entries.clone() {
18                  image.write_all(&(entry.name)).expect("Failed to write in
file!");
19                  image.write_all(&(mem::transmute:::<u16, [u8;
2] >(entry.start))).expect("Failed to write in file!");
20                  image.write_all(&(mem::transmute:::<u32, [u8;
4] >(entry.size))).expect("Failed to write in file!");
21              }
22              for entry in self.new_entries.clone() {
23                  // get filename from path
24                  let path = bytes_to_str(&entry.name);
25                  let filename = match Path::new(path).file_name() {
26                      Some(name) => name,
27                      None => return,
28                  };
29                  let filename = match filename.to_str() {

```

```

30         Some(name) => name,
31         None => return,
32     };
33     let mut raw_name : [u8;26] = [0;26];
34     let mut i = 0;
35     for byte in filename.bytes() {
36         raw_name[i] = byte;
37         i += 1;
38         if i == 26 { break; }
39     }
40
41     image.write_all(&(raw_name)).expect("Failed to write in
file!");
42     image.write_all(&(mem::transmute::<u16, [u8;
2]>(entry.start))).expect("Failed to write in file!");
43     image.write_all(&(mem::transmute::<u32, [u8;
4]>(entry.size))).expect("Failed to write in file!");
44
45     let mut file_buffer = Vec::new();
46     let s = bytes_to_str(&(entry.name));
47     let mut file = File::open(s).expect("File not found !");
48     file.read_to_end(&mut file_buffer).expect("Something went
wrong reading the file !");
49     let mut blocks = self.get_blocks(&mut entry.clone());
50     self.write_data(&mut blocks, file_buffer);
51     }
52 }
53 }
54 }

```

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

```

1 use super::*;
2
3 impl MicroFS {
4     pub fn fat_size(&mut self) -> usize {
5         self.sb.fat_size as usize
6     }
7
8     pub fn root_entry(&mut self) -> usize {
9         self.sb.root_entry as usize * self.sb.block_size as usize *
SECTOR_SIZE
10     }
11
12     pub fn entries_size(&mut self) -> usize {
13         self.sb.block_size as usize * SECTOR_SIZE
14     }
15 }

```

```

16     pub fn set_fat(&mut self) {
17         let mut file = File::open(self.image.clone()).expect("File not
found !");
18         let mut raw_fat = Vec::new();
19         file.read_to_end(&mut raw_fat).expect("Something went wrong
reading the file !");
20         for i in SECTOR_SIZE..(SECTOR_SIZE+self.fat_size()) {
21             self.fat.push(raw_fat[i]);
22         }
23     }
24
25     pub fn set_entries(&mut self) {
26         unsafe {
27             let mut file = File::open(self.image.clone()).expect("File
not found !");
28
29             let mut cnt = 0;
30             while cnt < self.entries_size() {
31                 let mut raw_name = [0;26];
32                 file.seek(SeekFrom::Start((self.root_entry() + cnt) as
u64)).expect("File seek failed !");
33                 file.read(&mut raw_name).expect("Something went wrong
reading the file !");
34                 if raw_name[0] != 0 {
35                     let mut raw_start = [0;2];
36                     file.read(&mut raw_start).expect("Something went
wrong reading the file !");
37                     let start = mem::transmute:::<[u8;2], u16>(raw_start);
38                     let mut raw_size = [0;4];
39                     file.read(&mut raw_size).expect("Something went wrong
reading the file !");
40                     let size = mem::transmute:::<[u8;4], u32>(raw_size);
41                     self.entries.push(Entry { name: raw_name, start:
start, size: size });
42                 }
43                 cnt += mem::size_of::<Entry>();
44             }
45         }
46     }
47
48     pub fn empty_blocks(&mut self, entry: &mut Entry) -> Vec<usize> {
49         let entries_blocks_size = self.entries_size() / (SECTOR_SIZE *
(self.sb.block_size as usize));
50         let data_start = entries_blocks_size + (self.sb.root_entry as
usize);
51
52         let mut cnt = 0;
53         let mut blocks = Vec::new();

```

```

54     for i in data_start..self.fat.len() {
55         if (self.fat)[i] == 0xff {
56             if cnt == 0 {
57                 entry.start = i as u16;
58             }
59             blocks.push(i);
60             cnt += 1;
61         }
62         if cnt >= (entry.size as usize / (SECTOR_SIZE *
self.sb.block_size as usize) + 1) {
63             break;
64         }
65     }
66     return blocks;
67 }
68
69 pub fn get_blocks(&mut self, entry: &mut Entry) -> Vec<usize> {
70     let mut blocks = Vec::new();
71     let mut block = entry.start as usize;
72     blocks.push(block);
73     loop {
74         block = self.fat[block] as usize;
75         if block != 0 {
76             blocks.push(block);
77         } else {
78             break;
79         }
80     }
81     return blocks;
82 }
83
84 pub fn update_fat(&mut self, blocks: &mut Vec<usize>, add: bool) {
85     for i in 0..blocks.len() {
86         if add {
87             if i == (blocks.len() - 1) {
88                 (self.fat)[blocks[i]] = 0;
89             } else {
90                 (self.fat)[blocks[i]] = blocks[i+1] as u8;
91             }
92         } else {
93             (self.fat)[blocks[i]] = 0xff;
94         }
95     }
96 }
97
98 pub fn write_data(&mut self, blocks: &mut Vec<usize>, data: Vec<u8>)
{

```

```

99         let mut file =
OpenOptions::new().read(true).write(true).open(self.image.clone()).expect("File
not found !");
100         let mut cnt = 0;
101         for block in blocks.iter() {
102             let offset = block * (self.sb.block_size as usize) *
SECTOR_SIZE;
103             file.seek(SeekFrom::Start(offset as u64)).expect("File seek
failed !");
104
105             let data_block_start = cnt * (self.sb.block_size as usize) *
SECTOR_SIZE;
106             let mut data_block_end = data_block_start +
(self.sb.block_size as usize) * SECTOR_SIZE;
107             if data_block_end > data.len() {
108                 data_block_end = data.len();
109             }
110
111             file.write_all(&(data[data_block_start..data_block_end])).expect("Failed
to write in file!");
112             cnt += 1;
113         }
114     }
115
116     pub fn bytes_to_str(bytes: &[u8]) -> &str {
117         let mut cnt = 0;
118         for &byte in bytes {
119             if byte == 0 {
120                 break;
121             }
122             cnt += 1;
123         }
124         str::from_utf8(&bytes[0..cnt]).expect("Found invalid UTF-8")
125     }

```

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

```

1  name: MicroFS
2  version: "0.1.0"
3  author: Orphée Antoniadis <orphee.antoniadis@gmail.com>
4  about: Tools to build a minimal file system
5  args:
6      - image:
7          help: Image of the file system
8          required: true
9          index: 1
10 subcommands:

```



```

11     - create:
12         about: Create the image of the filesystem
13         args:
14             - label:
15                 help: Label of the filesystem to create
16                 required: true
17                 index: 1
18             - block_size:
19                 help: Block size of the filesystem to create (in number of
sectors)
20                 required: true
21                 index: 2
22             - size:
23                 help: Size of the filesystem to create (in bytes)
24                 required: true
25                 index: 3
26
27     - add:
28         about: Add a file to the file system
29         args:
30             - file:
31                 help: File to add to the file system
32                 required: true
33                 index: 1
34
35     - del:
36         about: Remove a file from the file system
37         args:
38             - file:
39                 help: File to remove from the file system
40                 required: true
41                 index: 1
42
43     - list:
44         about: Display all the files of the file system
45
46     - info:
47         about: Display file system informations

```

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

```

1  #![crate_name = "micro_fs"]
2  #![crate_type = "lib"]
3
4  pub mod micro_fs;
5  pub use micro_fs::*;

```

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

```
1  #![crate_name = "micro_fs"]
2
3  use std::io;
4  use std::process;
5
6  #[macro_use]
7  extern crate clap;
8  use clap::App;
9
10 extern crate micro_fs;
11 use micro_fs::*;
12
13 fn main() {
14     let yaml = load_yaml!("cli.yml");
15     let matches = App::from_yaml(yaml).get_matches();
16
17     let image = matches.value_of("image").unwrap();
18     let mut fs = MicroFS::new(image);
19
20     match matches.subcommand() {
21         ("create", Some(create_matches)) => {
22             let label = create_matches.value_of("label").unwrap();
23             let bs = value_t!(create_matches.value_of("block_size"),
u8).unwrap_or_else(|e| e.exit());
24             let size = value_t!(create_matches.value_of("size"),
usize).unwrap_or_else(|e| e.exit());
25             fs.create(label, bs, size);
26         },
27         ("add", Some(add_matches)) => {
28             fs.add(add_matches.value_of("file").unwrap());
29             fs.save();
30         },
31         ("del", Some(del_matches)) => {
32             fs.del(del_matches.value_of("file").unwrap());
33             fs.save();
34         },
35         ("list", Some(_matches)) => fs.list(),
36         ("info", Some(_matches)) => fs.info(),
37         ("", None) => {
38             loop {
39                 let mut choice = String::new();
40
41                 println!("\nMENU\n");
42                 println!("0: quit");
43                 println!("1: create <label> <block_size> <fs_size>");
44                 println!("2: add <file>");
45                 println!("3: del <file>");
```

```

46     println!("4: list");
47     println!("5: info");
48     println!("6: save");
49
50     io::stdin().read_line(&mut choice).expect("Failed to read
line !");
51     let choice : u8 = match choice.trim().parse() {
52         Ok(num) => num,
53         Err(_) => {
54             println!("Not a number !");
55             continue;
56         }
57     };
58
59     match choice {
60         0 => break,
61         1 => {
62             println!("\n[0] Label :");
63             let mut label = String::new();
64             io::stdin().read_line(&mut label).expect("Failed
to read line !");
65             label = label.trim().to_string();
66             println!("[1] Block size :");
67             let mut str_bs = String::new();
68             io::stdin().read_line(&mut str_bs).expect("Failed
to read line !");
69             let bs = match str_bs.trim().parse() {
70                 Ok(num) => num,
71                 Err(_) => {
72                     println!("Not a number !");
73                     continue;
74                 }
75             };
76             println!("[2] FS size :");
77             let mut str_size = String::new();
78             io::stdin().read_line(&mut
str_size).expect("Failed to read line !");
79             let size = match str_size.trim().parse() {
80                 Ok(num) => num,
81                 Err(_) => {
82                     println!("Not a number !");
83                     continue;
84                 }
85             };
86             println!("");
87             fs.create(&label, bs, size);
88         },
89         2 => {

```

```

90         println!("\n[0] File :");
91         let mut filename = String::new();
92         io::stdin().read_line(&mut
filename).expect("Failed to read line !");
93         filename = filename.trim().to_string();
94         println!("");
95         fs.add(&filename);
96     },
97     3 => {
98         println!("\n[0] File :");
99         let mut filename = String::new();
100        io::stdin().read_line(&mut
filename).expect("Failed to read line !");
101        filename = filename.trim().to_string();
102        println!("");
103        fs.del(&filename);
104    },
105    4 => fs.list(),
106    5 => fs.info(),
107    6 => fs.save(),
108    _ => println!("Choice {} does not exist", choice),
109    }
110    },
111    },
112    -        => unreachable!(),
113    }
114    process::exit(0);
115    }

```

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

```

1  extern crate micro_fs;
2  use micro_fs::*;
3  use micro_fs::utils::bytes_to_str;
4
5  use std::mem;
6  use std::fs;
7  use std::fs::File;
8  use std::io::prelude::*;
9  use std::io::SeekFrom;
10
11  #[test]
12  fn constructors() {
13      let mut test_fs = MicroFS::new("fs1_test.img");
14      test_fs.sb = SuperBlock::new("test", 1, 100000);
15      assert_eq!(test_fs.image, "fs1_test.img");
16      assert_eq!(test_fs.sb.sector_size, SECTOR_SIZE as u16);
17      assert_eq!(test_fs.sb.block_size, 1);

```

```

18     assert_eq!(test_fs.sb.fat_size, 195);
19     assert_eq!(test_fs.sb.version, 1);
20     assert_eq!(test_fs.sb.root_entry, 2);
21     assert_eq!(test_fs.sb.label, [b't', b'e', b's', b't', b'\0', b'\0',
b'\0', b'\0']);
22     assert_eq!(test_fs.sb.signature, MAGIC);
23 }
24
25 #[test]
26 fn create() {
27     let mut test_fs = MicroFS::new("fs1_test.img");
28     test_fs.create("test", 1, 100000);
29     let mut raw_sb = [0; SECTOR_SIZE];
30     let mut file = File::open(test_fs.image.clone()).expect("File not
found !");
31     file.read(&mut raw_sb).expect("Something went wrong reading the file
!");
32
33     let signature = unsafe { mem::transmute:::<[u8; 2], u16>([raw_sb[510],
raw_sb[511]]) };
34     assert_eq!(signature, MAGIC);
35     let bs = raw_sb[13];
36     assert_eq!(bs, 1);
37     let raw_label = &raw_sb[82..90];
38     let label = bytes_to_str(raw_label);
39     assert_eq!(label, "test");
40     let size = fs::metadata(test_fs.image.clone()).expect("Failed
getting metadata!").len() as usize;
41     assert_eq!(size, 99840);
42
43     let mut raw_fat = [0; 195];
44     file.read(&mut raw_fat).expect("Something went wrong reading the file
!");
45     assert_eq!(&raw_fat[..], &[0xff; 195][..]);
46
47     file.seek(SeekFrom::Start(1024)).expect("File seek failed !");
48     let mut raw_entries = [0; 512];
49     file.read(&mut raw_entries).expect("Something went wrong reading the
file !");
50     assert_eq!(&raw_entries[..], &[0; 512][..]);
51
52     fs::remove_file("fs1_test.img").expect("Failed removing the file");
53 }
54
55 #[test]
56 fn add() {
57     let mut test_fs = MicroFS::new("fs1_test.img");
58     test_fs.create("test", 1, 100000);

```

```

59     test_fs.add("tests/test1.txt");
60
61     assert_eq!(bytes_to_str(&test_fs.new_entries[0].name),
"tests/test1.txt");
62     assert_eq!(test_fs.new_entries[0].start, 3);
63     assert_eq!(test_fs.new_entries[0].size,
fs::metadata("tests/test1.txt").expect("Failed getting metadata!").len()
as u32);
64
65     fs::remove_file("fs1_test.img").expect("Failed removing the file");
66 }
67
68 #[test]
69 fn save() {
70     let mut cmp_fs = MicroFS {
71         image: "fs1_test.img".to_string(),
72         sb: SuperBlock::new("test", 1, 100000),
73         fat: Vec::new(),
74         entries: Vec::new(),
75         new_entries: Vec::new()
76     };
77
78     let mut test_fs = MicroFS::new("fs1_test.img");
79     test_fs.create("test", 1, 100000);
80     test_fs.add("tests/test1.txt");
81     test_fs.save();
82
83     cmp_fs.set_fat();
84     assert_eq!(test_fs.fat, cmp_fs.fat);
85
86     cmp_fs.set_entries();
87     assert_eq!(test_fs.new_entries[0].start, cmp_fs.entries[0].start);
88     assert_eq!(test_fs.new_entries[0].size, cmp_fs.entries[0].size);
89
90     let mut raw_data = [0; SECTOR_SIZE];
91     let mut file = File::open(test_fs.image.clone()).expect("File not
found !");
92     file.seek(SeekFrom::Start(1536)).expect("File seek failed !");
93     file.read(&mut raw_data).expect("Something went wrong reading the
file !");
94     for i in 0..SECTOR_SIZE {
95         assert_eq!(raw_data[i], TEST1[i]);
96     }
97
98     fs::remove_file("fs1_test.img").expect("Failed removing the file");
99 }
100
101 #[test]

```

```

102 fn remove() {
103     let mut cmp_fs = MicroFS {
104         image: "fs1_test.img".to_string(),
105         sb: SuperBlock::new("test", 1, 100000),
106         fat: Vec::new(),
107         entries: Vec::new(),
108         new_entries: Vec::new()
109     };
110
111     let mut test_fs = MicroFS::new("fs1_test.img");
112     test_fs.create("test", 1, 100000);
113     test_fs.add("tests/test1.txt");
114     test_fs.save();
115     test_fs.del("tests/test1.txt");
116     test_fs.save();
117
118     cmp_fs.set_fat();
119     assert_eq!(&cmp_fs.fat[..], &[0xff; 195][..]);
120
121     cmp_fs.set_entries();
122     assert_eq!(cmp_fs.new_entries.len(), 0);
123
124     fs::remove_file("fs1_test.img").expect("Failed removing the file");
125 }
126
127 #[test]
128 fn multiple_blocks() {
129     let mut test_fs = MicroFS::new("fs1_test.img");
130     test_fs.create("test", 1, 100000);
131     test_fs.add("tests/test1.txt");
132     test_fs.add("tests/test2.txt");
133     test_fs.save();
134
135     let mut raw_data = [0; SECTOR_SIZE*5];
136     let mut file = File::open(test_fs.image.clone()).expect("File not
found !");
137     file.seek(SeekFrom::Start(2048)).expect("File seek failed !");
138     file.read(&mut raw_data).expect("Something went wrong reading the
file !");
139
140     assert_eq!(&raw_data[..], &TEST2[..]);
141     fs::remove_file("fs1_test.img").expect("Failed removing the file");
142 }
143
144 const TEST1 : [u8;SECTOR_SIZE] = [
145     0x4c, 0x6f, 0x72, 0x65, 0x6d, 0x20, 0x69, 0x70,
146     0x73, 0x75, 0x6d, 0x20, 0x64, 0x6f, 0x6c, 0x6f,
147     0x72, 0x20, 0x73, 0x69, 0x74, 0x20, 0x61, 0x6d,

```

148	0x65, 0x74, 0x2c, 0x20, 0x63, 0x6f, 0x6e, 0x73,
149	0x65, 0x63, 0x74, 0x65, 0x74, 0x75, 0x72, 0x20,
150	0x61, 0x64, 0x69, 0x70, 0x69, 0x73, 0x69, 0x63,
151	0x69, 0x6e, 0x67, 0x20, 0x65, 0x6c, 0x69, 0x74,
152	0x2c, 0x20, 0x73, 0x65, 0x64, 0x20, 0x64, 0x6f,
153	0x20, 0x65, 0x69, 0x75, 0x73, 0x6d, 0x6f, 0x64,
154	0x20, 0x74, 0x65, 0x6d, 0x70, 0x6f, 0x72, 0x20,
155	0x69, 0x6e, 0x63, 0x69, 0x64, 0x69, 0x64, 0x75,
156	0x6e, 0x74, 0x20, 0x75, 0x74, 0x20, 0x6c, 0x61,
157	0x62, 0x6f, 0x72, 0x65, 0x20, 0x65, 0x74, 0x20,
158	0x64, 0x6f, 0x6c, 0x6f, 0x72, 0x65, 0x20, 0x6d,
159	0x61, 0x67, 0x6e, 0x61, 0x20, 0x61, 0x6c, 0x69,
160	0x71, 0x75, 0x61, 0x2e, 0x20, 0x55, 0x74, 0x20,
161	0x65, 0x6e, 0x69, 0x6d, 0x20, 0x61, 0x64, 0x20,
162	0x6d, 0x69, 0x6e, 0x69, 0x6d, 0x20, 0x76, 0x65,
163	0x6e, 0x69, 0x61, 0x6d, 0x2c, 0x20, 0x71, 0x75,
164	0x69, 0x73, 0x20, 0x6e, 0x6f, 0x73, 0x74, 0x72,
165	0x75, 0x64, 0x20, 0x65, 0x78, 0x65, 0x72, 0x63,
166	0x69, 0x74, 0x61, 0x74, 0x69, 0x6f, 0x6e, 0x20,
167	0x75, 0x6c, 0x6c, 0x61, 0x6d, 0x63, 0x6f, 0x20,
168	0x6c, 0x61, 0x62, 0x6f, 0x72, 0x69, 0x73, 0x20,
169	0x6e, 0x69, 0x73, 0x69, 0x20, 0x75, 0x74, 0x20,
170	0x61, 0x6c, 0x69, 0x71, 0x75, 0x69, 0x70, 0x20,
171	0x65, 0x78, 0x20, 0x65, 0x61, 0x20, 0x63, 0x6f,
172	0x6d, 0x6d, 0x6f, 0x64, 0x6f, 0x20, 0x63, 0x6f,
173	0x6e, 0x73, 0x65, 0x71, 0x75, 0x61, 0x74, 0x2e,
174	0x20, 0x44, 0x75, 0x69, 0x73, 0x20, 0x61, 0x75,
175	0x74, 0x65, 0x20, 0x69, 0x72, 0x75, 0x72, 0x65,
176	0x20, 0x64, 0x6f, 0x6c, 0x6f, 0x72, 0x20, 0x69,
177	0x6e, 0x20, 0x72, 0x65, 0x70, 0x72, 0x65, 0x68,
178	0x65, 0x6e, 0x64, 0x65, 0x72, 0x69, 0x74, 0x20,
179	0x69, 0x6e, 0x20, 0x76, 0x6f, 0x6c, 0x75, 0x70,
180	0x74, 0x61, 0x74, 0x65, 0x20, 0x76, 0x65, 0x6c,
181	0x69, 0x74, 0x20, 0x65, 0x73, 0x73, 0x65, 0x20,
182	0x63, 0x69, 0x6c, 0x6c, 0x75, 0x6d, 0x20, 0x64,
183	0x6f, 0x6c, 0x6f, 0x72, 0x65, 0x20, 0x65, 0x75,
184	0x20, 0x66, 0x75, 0x67, 0x69, 0x61, 0x74, 0x20,
185	0x6e, 0x75, 0x6c, 0x6c, 0x61, 0x20, 0x70, 0x61,
186	0x72, 0x69, 0x61, 0x74, 0x75, 0x72, 0x2e, 0x20,
187	0x45, 0x78, 0x63, 0x65, 0x70, 0x74, 0x65, 0x75,
188	0x72, 0x20, 0x73, 0x69, 0x6e, 0x74, 0x20, 0x6f,
189	0x63, 0x63, 0x61, 0x65, 0x63, 0x61, 0x74, 0x20,
190	0x63, 0x75, 0x70, 0x69, 0x64, 0x61, 0x74, 0x61,
191	0x74, 0x20, 0x6e, 0x6f, 0x6e, 0x20, 0x70, 0x72,
192	0x6f, 0x69, 0x64, 0x65, 0x6e, 0x74, 0x2c, 0x20,
193	0x73, 0x75, 0x6e, 0x74, 0x20, 0x69, 0x6e, 0x20,
194	0x63, 0x75, 0x6c, 0x70, 0x61, 0x20, 0x71, 0x75,
195	0x69, 0x20, 0x6f, 0x66, 0x66, 0x69, 0x63, 0x69,


```

196     0x61, 0x20, 0x64, 0x65, 0x73, 0x65, 0x72, 0x75,
197     0x6e, 0x74, 0x20, 0x6d, 0x6f, 0x6c, 0x6c, 0x69,
198     0x74, 0x20, 0x61, 0x6e, 0x69, 0x6d, 0x20, 0x69,
199     0x64, 0x20, 0x65, 0x73, 0x74, 0x20, 0x6c, 0x61,
200     0x62, 0x6f, 0x72, 0x75, 0x6d, 0x2e, 0x0, 0x0,
201     0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
202     0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
203     0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
204     0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
205     0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
206     0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
207     0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
208     0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
209 ];
210
211 const TEST2 : [u8;SECTOR_SIZE*5] = [
212     0x4c, 0x6f, 0x72, 0x65, 0x6d, 0x20, 0x69, 0x70,
213     0x73, 0x75, 0x6d, 0x20, 0x64, 0x6f, 0x6c, 0x6f,
214     0x72, 0x20, 0x73, 0x69, 0x74, 0x20, 0x61, 0x6d,
215     0x65, 0x74, 0x2c, 0x20, 0x63, 0x6f, 0x6e, 0x73,
216     0x65, 0x63, 0x74, 0x65, 0x74, 0x75, 0x72, 0x20,
217     0x61, 0x64, 0x69, 0x70, 0x69, 0x73, 0x69, 0x63,
218     0x69, 0x6e, 0x67, 0x20, 0x65, 0x6c, 0x69, 0x74,
219     0x2c, 0x20, 0x73, 0x65, 0x64, 0x20, 0x64, 0x6f,
220     0x20, 0x65, 0x69, 0x75, 0x73, 0x6d, 0x6f, 0x64,
221     0x20, 0x74, 0x65, 0x6d, 0x70, 0x6f, 0x72, 0x20,
222     0x69, 0x6e, 0x63, 0x69, 0x64, 0x69, 0x64, 0x75,
223     0x6e, 0x74, 0x20, 0x75, 0x74, 0x20, 0x6c, 0x61,
224     0x62, 0x6f, 0x72, 0x65, 0x20, 0x65, 0x74, 0x20,
225     0x64, 0x6f, 0x6c, 0x6f, 0x72, 0x65, 0x20, 0x6d,
226     0x61, 0x67, 0x6e, 0x61, 0x20, 0x61, 0x6c, 0x69,
227     0x71, 0x75, 0x61, 0x2e, 0x20, 0x55, 0x74, 0x20,
228     0x65, 0x6e, 0x69, 0x6d, 0x20, 0x61, 0x64, 0x20,
229     0x6d, 0x69, 0x6e, 0x69, 0x6d, 0x20, 0x76, 0x65,
230     0x6e, 0x69, 0x61, 0x6d, 0x2c, 0x20, 0x71, 0x75,
231     0x69, 0x73, 0x20, 0x6e, 0x6f, 0x73, 0x74, 0x72,
232     0x75, 0x64, 0x20, 0x65, 0x78, 0x65, 0x72, 0x63,
233     0x69, 0x74, 0x61, 0x74, 0x69, 0x6f, 0x6e, 0x20,
234     0x75, 0x6c, 0x6c, 0x61, 0x6d, 0x63, 0x6f, 0x20,
235     0x6c, 0x61, 0x62, 0x6f, 0x72, 0x69, 0x73, 0x20,
236     0x6e, 0x69, 0x73, 0x69, 0x20, 0x75, 0x74, 0x20,
237     0x61, 0x6c, 0x69, 0x71, 0x75, 0x69, 0x70, 0x20,
238     0x65, 0x78, 0x20, 0x65, 0x61, 0x20, 0x63, 0x6f,
239     0x6d, 0x6d, 0x6f, 0x64, 0x6f, 0x20, 0x63, 0x6f,
240     0x6e, 0x73, 0x65, 0x71, 0x75, 0x61, 0x74, 0x2e,
241     0x20, 0x44, 0x75, 0x69, 0x73, 0x20, 0x61, 0x75,
242     0x74, 0x65, 0x20, 0x69, 0x72, 0x75, 0x72, 0x65,
243     0x20, 0x64, 0x6f, 0x6c, 0x6f, 0x72, 0x20, 0x69,

```

244	0x6e, 0x20, 0x72, 0x65, 0x70, 0x72, 0x65, 0x68,
245	0x65, 0x6e, 0x64, 0x65, 0x72, 0x69, 0x74, 0x20,
246	0x69, 0x6e, 0x20, 0x76, 0x6f, 0x6c, 0x75, 0x70,
247	0x74, 0x61, 0x74, 0x65, 0x20, 0x76, 0x65, 0x6c,
248	0x69, 0x74, 0x20, 0x65, 0x73, 0x73, 0x65, 0x20,
249	0x63, 0x69, 0x6c, 0x6c, 0x75, 0x6d, 0x20, 0x64,
250	0x6f, 0x6c, 0x6f, 0x72, 0x65, 0x20, 0x65, 0x75,
251	0x20, 0x66, 0x75, 0x67, 0x69, 0x61, 0x74, 0x20,
252	0x6e, 0x75, 0x6c, 0x6c, 0x61, 0x20, 0x70, 0x61,
253	0x72, 0x69, 0x61, 0x74, 0x75, 0x72, 0x2e, 0x20,
254	0x45, 0x78, 0x63, 0x65, 0x70, 0x74, 0x65, 0x75,
255	0x72, 0x20, 0x73, 0x69, 0x6e, 0x74, 0x20, 0x6f,
256	0x63, 0x63, 0x61, 0x65, 0x63, 0x61, 0x74, 0x20,
257	0x63, 0x75, 0x70, 0x69, 0x64, 0x61, 0x74, 0x61,
258	0x74, 0x20, 0x6e, 0x6f, 0x6e, 0x20, 0x70, 0x72,
259	0x6f, 0x69, 0x64, 0x65, 0x6e, 0x74, 0x2c, 0x20,
260	0x73, 0x75, 0x6e, 0x74, 0x20, 0x69, 0x6e, 0x20,
261	0x63, 0x75, 0x6c, 0x70, 0x61, 0x20, 0x71, 0x75,
262	0x69, 0x20, 0x6f, 0x66, 0x66, 0x69, 0x63, 0x69,
263	0x61, 0x20, 0x64, 0x65, 0x73, 0x65, 0x72, 0x75,
264	0x6e, 0x74, 0x20, 0x6d, 0x6f, 0x6c, 0x6c, 0x69,
265	0x74, 0x20, 0x61, 0x6e, 0x69, 0x6d, 0x20, 0x69,
266	0x64, 0x20, 0x65, 0x73, 0x74, 0x20, 0x6c, 0x61,
267	0x62, 0x6f, 0x72, 0x75, 0x6d, 0x2e, 0xa, 0x4c,
268	0x6f, 0x72, 0x65, 0x6d, 0x20, 0x69, 0x70, 0x73,
269	0x75, 0x6d, 0x20, 0x64, 0x6f, 0x6c, 0x6f, 0x72,
270	0x20, 0x73, 0x69, 0x74, 0x20, 0x61, 0x6d, 0x65,
271	0x74, 0x2c, 0x20, 0x63, 0x6f, 0x6e, 0x73, 0x65,
272	0x63, 0x74, 0x65, 0x74, 0x75, 0x72, 0x20, 0x61,
273	0x64, 0x69, 0x70, 0x69, 0x73, 0x69, 0x63, 0x69,
274	0x6e, 0x67, 0x20, 0x65, 0x6c, 0x69, 0x74, 0x2c,
275	0x20, 0x73, 0x65, 0x64, 0x20, 0x64, 0x6f, 0x20,
276	0x65, 0x69, 0x75, 0x73, 0x6d, 0x6f, 0x64, 0x20,
277	0x74, 0x65, 0x6d, 0x70, 0x6f, 0x72, 0x20, 0x69,
278	0x6e, 0x63, 0x69, 0x64, 0x69, 0x64, 0x75, 0x6e,
279	0x74, 0x20, 0x75, 0x74, 0x20, 0x6c, 0x61, 0x62,
280	0x6f, 0x72, 0x65, 0x20, 0x65, 0x74, 0x20, 0x64,
281	0x6f, 0x6c, 0x6f, 0x72, 0x65, 0x20, 0x6d, 0x61,
282	0x67, 0x6e, 0x61, 0x20, 0x61, 0x6c, 0x69, 0x71,
283	0x75, 0x61, 0x2e, 0x20, 0x55, 0x74, 0x20, 0x65,
284	0x6e, 0x69, 0x6d, 0x20, 0x61, 0x64, 0x20, 0x6d,
285	0x69, 0x6e, 0x69, 0x6d, 0x20, 0x76, 0x65, 0x6e,
286	0x69, 0x61, 0x6d, 0x2c, 0x20, 0x71, 0x75, 0x69,
287	0x73, 0x20, 0x6e, 0x6f, 0x73, 0x74, 0x72, 0x75,
288	0x64, 0x20, 0x65, 0x78, 0x65, 0x72, 0x63, 0x69,
289	0x74, 0x61, 0x74, 0x69, 0x6f, 0x6e, 0x20, 0x75,
290	0x6c, 0x6c, 0x61, 0x6d, 0x63, 0x6f, 0x20, 0x6c,
291	0x61, 0x62, 0x6f, 0x72, 0x69, 0x73, 0x20, 0x6e,

292	0x69, 0x73, 0x69, 0x20, 0x75, 0x74, 0x20, 0x61,
293	0x6c, 0x69, 0x71, 0x75, 0x69, 0x70, 0x20, 0x65,
294	0x78, 0x20, 0x65, 0x61, 0x20, 0x63, 0x6f, 0x6d,
295	0x6d, 0x6f, 0x64, 0x6f, 0x20, 0x63, 0x6f, 0x6e,
296	0x73, 0x65, 0x71, 0x75, 0x61, 0x74, 0x2e, 0x20,
297	0x44, 0x75, 0x69, 0x73, 0x20, 0x61, 0x75, 0x74,
298	0x65, 0x20, 0x69, 0x72, 0x75, 0x72, 0x65, 0x20,
299	0x64, 0x6f, 0x6c, 0x6f, 0x72, 0x20, 0x69, 0x6e,
300	0x20, 0x72, 0x65, 0x70, 0x72, 0x65, 0x68, 0x65,
301	0x6e, 0x64, 0x65, 0x72, 0x69, 0x74, 0x20, 0x69,
302	0x6e, 0x20, 0x76, 0x6f, 0x6c, 0x75, 0x70, 0x74,
303	0x61, 0x74, 0x65, 0x20, 0x76, 0x65, 0x6c, 0x69,
304	0x74, 0x20, 0x65, 0x73, 0x73, 0x65, 0x20, 0x63,
305	0x69, 0x6c, 0x6c, 0x75, 0x6d, 0x20, 0x64, 0x6f,
306	0x6c, 0x6f, 0x72, 0x65, 0x20, 0x65, 0x75, 0x20,
307	0x66, 0x75, 0x67, 0x69, 0x61, 0x74, 0x20, 0x6e,
308	0x75, 0x6c, 0x6c, 0x61, 0x20, 0x70, 0x61, 0x72,
309	0x69, 0x61, 0x74, 0x75, 0x72, 0x2e, 0x20, 0x45,
310	0x78, 0x63, 0x65, 0x70, 0x74, 0x65, 0x75, 0x72,
311	0x20, 0x73, 0x69, 0x6e, 0x74, 0x20, 0x6f, 0x63,
312	0x63, 0x61, 0x65, 0x63, 0x61, 0x74, 0x20, 0x63,
313	0x75, 0x70, 0x69, 0x64, 0x61, 0x74, 0x61, 0x74,
314	0x20, 0x6e, 0x6f, 0x6e, 0x20, 0x70, 0x72, 0x6f,
315	0x69, 0x64, 0x65, 0x6e, 0x74, 0x2c, 0x20, 0x73,
316	0x75, 0x6e, 0x74, 0x20, 0x69, 0x6e, 0x20, 0x63,
317	0x75, 0x6c, 0x70, 0x61, 0x20, 0x71, 0x75, 0x69,
318	0x20, 0x6f, 0x66, 0x66, 0x69, 0x63, 0x69, 0x61,
319	0x20, 0x64, 0x65, 0x73, 0x65, 0x72, 0x75, 0x6e,
320	0x74, 0x20, 0x6d, 0x6f, 0x6c, 0x6c, 0x69, 0x74,
321	0x20, 0x61, 0x6e, 0x69, 0x6d, 0x20, 0x69, 0x64,
322	0x20, 0x65, 0x73, 0x74, 0x20, 0x6c, 0x61, 0x62,
323	0x6f, 0x72, 0x75, 0x6d, 0x2e, 0xa, 0x4c, 0x6f,
324	0x72, 0x65, 0x6d, 0x20, 0x69, 0x70, 0x73, 0x75,
325	0x6d, 0x20, 0x64, 0x6f, 0x6c, 0x6f, 0x72, 0x20,
326	0x73, 0x69, 0x74, 0x20, 0x61, 0x6d, 0x65, 0x74,
327	0x2c, 0x20, 0x63, 0x6f, 0x6e, 0x73, 0x65, 0x63,
328	0x74, 0x65, 0x74, 0x75, 0x72, 0x20, 0x61, 0x64,
329	0x69, 0x70, 0x69, 0x73, 0x69, 0x63, 0x69, 0x6e,
330	0x67, 0x20, 0x65, 0x6c, 0x69, 0x74, 0x2c, 0x20,
331	0x73, 0x65, 0x64, 0x20, 0x64, 0x6f, 0x20, 0x65,
332	0x69, 0x75, 0x73, 0x6d, 0x6f, 0x64, 0x20, 0x74,
333	0x65, 0x6d, 0x70, 0x6f, 0x72, 0x20, 0x69, 0x6e,
334	0x63, 0x69, 0x64, 0x69, 0x64, 0x75, 0x6e, 0x74,
335	0x20, 0x75, 0x74, 0x20, 0x6c, 0x61, 0x62, 0x6f,
336	0x72, 0x65, 0x20, 0x65, 0x74, 0x20, 0x64, 0x6f,
337	0x6c, 0x6f, 0x72, 0x65, 0x20, 0x6d, 0x61, 0x67,
338	0x6e, 0x61, 0x20, 0x61, 0x6c, 0x69, 0x71, 0x75,
339	0x61, 0x2e, 0x20, 0x55, 0x74, 0x20, 0x65, 0x6e,

340	0x69, 0x6d, 0x20, 0x61, 0x64, 0x20, 0x6d, 0x69,
341	0x6e, 0x69, 0x6d, 0x20, 0x76, 0x65, 0x6e, 0x69,
342	0x61, 0x6d, 0x2c, 0x20, 0x71, 0x75, 0x69, 0x73,
343	0x20, 0x6e, 0x6f, 0x73, 0x74, 0x72, 0x75, 0x64,
344	0x20, 0x65, 0x78, 0x65, 0x72, 0x63, 0x69, 0x74,
345	0x61, 0x74, 0x69, 0x6f, 0x6e, 0x20, 0x75, 0x6c,
346	0x6c, 0x61, 0x6d, 0x63, 0x6f, 0x20, 0x6c, 0x61,
347	0x62, 0x6f, 0x72, 0x69, 0x73, 0x20, 0x6e, 0x69,
348	0x73, 0x69, 0x20, 0x75, 0x74, 0x20, 0x61, 0x6c,
349	0x69, 0x71, 0x75, 0x69, 0x70, 0x20, 0x65, 0x78,
350	0x20, 0x65, 0x61, 0x20, 0x63, 0x6f, 0x6d, 0x6d,
351	0x6f, 0x64, 0x6f, 0x20, 0x63, 0x6f, 0x6e, 0x73,
352	0x65, 0x71, 0x75, 0x61, 0x74, 0x2e, 0x20, 0x44,
353	0x75, 0x69, 0x73, 0x20, 0x61, 0x75, 0x74, 0x65,
354	0x20, 0x69, 0x72, 0x75, 0x72, 0x65, 0x20, 0x64,
355	0x6f, 0x6c, 0x6f, 0x72, 0x20, 0x69, 0x6e, 0x20,
356	0x72, 0x65, 0x70, 0x72, 0x65, 0x68, 0x65, 0x6e,
357	0x64, 0x65, 0x72, 0x69, 0x74, 0x20, 0x69, 0x6e,
358	0x20, 0x76, 0x6f, 0x6c, 0x75, 0x70, 0x74, 0x61,
359	0x74, 0x65, 0x20, 0x76, 0x65, 0x6c, 0x69, 0x74,
360	0x20, 0x65, 0x73, 0x73, 0x65, 0x20, 0x63, 0x69,
361	0x6c, 0x6c, 0x75, 0x6d, 0x20, 0x64, 0x6f, 0x6c,
362	0x6f, 0x72, 0x65, 0x20, 0x65, 0x75, 0x20, 0x66,
363	0x75, 0x67, 0x69, 0x61, 0x74, 0x20, 0x6e, 0x75,
364	0x6c, 0x6c, 0x61, 0x20, 0x70, 0x61, 0x72, 0x69,
365	0x61, 0x74, 0x75, 0x72, 0x2e, 0x20, 0x45, 0x78,
366	0x63, 0x65, 0x70, 0x74, 0x65, 0x75, 0x72, 0x20,
367	0x73, 0x69, 0x6e, 0x74, 0x20, 0x6f, 0x63, 0x63,
368	0x61, 0x65, 0x63, 0x61, 0x74, 0x20, 0x63, 0x75,
369	0x70, 0x69, 0x64, 0x61, 0x74, 0x61, 0x74, 0x20,
370	0x6e, 0x6f, 0x6e, 0x20, 0x70, 0x72, 0x6f, 0x69,
371	0x64, 0x65, 0x6e, 0x74, 0x2c, 0x20, 0x73, 0x75,
372	0x6e, 0x74, 0x20, 0x69, 0x6e, 0x20, 0x63, 0x75,
373	0x6c, 0x70, 0x61, 0x20, 0x71, 0x75, 0x69, 0x20,
374	0x6f, 0x66, 0x66, 0x69, 0x63, 0x69, 0x61, 0x20,
375	0x64, 0x65, 0x73, 0x65, 0x72, 0x75, 0x6e, 0x74,
376	0x20, 0x6d, 0x6f, 0x6c, 0x6c, 0x69, 0x74, 0x20,
377	0x61, 0x6e, 0x69, 0x6d, 0x20, 0x69, 0x64, 0x20,
378	0x65, 0x73, 0x74, 0x20, 0x6c, 0x61, 0x62, 0x6f,
379	0x72, 0x75, 0x6d, 0x2e, 0xa, 0x4c, 0x6f, 0x72,
380	0x65, 0x6d, 0x20, 0x69, 0x70, 0x73, 0x75, 0x6d,
381	0x20, 0x64, 0x6f, 0x6c, 0x6f, 0x72, 0x20, 0x73,
382	0x69, 0x74, 0x20, 0x61, 0x6d, 0x65, 0x74, 0x2c,
383	0x20, 0x63, 0x6f, 0x6e, 0x73, 0x65, 0x63, 0x74,
384	0x65, 0x74, 0x75, 0x72, 0x20, 0x61, 0x64, 0x69,
385	0x70, 0x69, 0x73, 0x69, 0x63, 0x69, 0x6e, 0x67,
386	0x20, 0x65, 0x6c, 0x69, 0x74, 0x2c, 0x20, 0x73,
387	0x65, 0x64, 0x20, 0x64, 0x6f, 0x20, 0x65, 0x69,

388	0x75, 0x73, 0x6d, 0x6f, 0x64, 0x20, 0x74, 0x65,
389	0x6d, 0x70, 0x6f, 0x72, 0x20, 0x69, 0x6e, 0x63,
390	0x69, 0x64, 0x69, 0x64, 0x75, 0x6e, 0x74, 0x20,
391	0x75, 0x74, 0x20, 0x6c, 0x61, 0x62, 0x6f, 0x72,
392	0x65, 0x20, 0x65, 0x74, 0x20, 0x64, 0x6f, 0x6c,
393	0x6f, 0x72, 0x65, 0x20, 0x6d, 0x61, 0x67, 0x6e,
394	0x61, 0x20, 0x61, 0x6c, 0x69, 0x71, 0x75, 0x61,
395	0x2e, 0x20, 0x55, 0x74, 0x20, 0x65, 0x6e, 0x69,
396	0x6d, 0x20, 0x61, 0x64, 0x20, 0x6d, 0x69, 0x6e,
397	0x69, 0x6d, 0x20, 0x76, 0x65, 0x6e, 0x69, 0x61,
398	0x6d, 0x2c, 0x20, 0x71, 0x75, 0x69, 0x73, 0x20,
399	0x6e, 0x6f, 0x73, 0x74, 0x72, 0x75, 0x64, 0x20,
400	0x65, 0x78, 0x65, 0x72, 0x63, 0x69, 0x74, 0x61,
401	0x74, 0x69, 0x6f, 0x6e, 0x20, 0x75, 0x6c, 0x6c,
402	0x61, 0x6d, 0x63, 0x6f, 0x20, 0x6c, 0x61, 0x62,
403	0x6f, 0x72, 0x69, 0x73, 0x20, 0x6e, 0x69, 0x73,
404	0x69, 0x20, 0x75, 0x74, 0x20, 0x61, 0x6c, 0x69,
405	0x71, 0x75, 0x69, 0x70, 0x20, 0x65, 0x78, 0x20,
406	0x65, 0x61, 0x20, 0x63, 0x6f, 0x6d, 0x6d, 0x6f,
407	0x64, 0x6f, 0x20, 0x63, 0x6f, 0x6e, 0x73, 0x65,
408	0x71, 0x75, 0x61, 0x74, 0x2e, 0x20, 0x44, 0x75,
409	0x69, 0x73, 0x20, 0x61, 0x75, 0x74, 0x65, 0x20,
410	0x69, 0x72, 0x75, 0x72, 0x65, 0x20, 0x64, 0x6f,
411	0x6c, 0x6f, 0x72, 0x20, 0x69, 0x6e, 0x20, 0x72,
412	0x65, 0x70, 0x72, 0x65, 0x68, 0x65, 0x6e, 0x64,
413	0x65, 0x72, 0x69, 0x74, 0x20, 0x69, 0x6e, 0x20,
414	0x76, 0x6f, 0x6c, 0x75, 0x70, 0x74, 0x61, 0x74,
415	0x65, 0x20, 0x76, 0x65, 0x6c, 0x69, 0x74, 0x20,
416	0x65, 0x73, 0x73, 0x65, 0x20, 0x63, 0x69, 0x6c,
417	0x6c, 0x75, 0x6d, 0x20, 0x64, 0x6f, 0x6c, 0x6f,
418	0x72, 0x65, 0x20, 0x65, 0x75, 0x20, 0x66, 0x75,
419	0x67, 0x69, 0x61, 0x74, 0x20, 0x6e, 0x75, 0x6c,
420	0x6c, 0x61, 0x20, 0x70, 0x61, 0x72, 0x69, 0x61,
421	0x74, 0x75, 0x72, 0x2e, 0x20, 0x45, 0x78, 0x63,
422	0x65, 0x70, 0x74, 0x65, 0x75, 0x72, 0x20, 0x73,
423	0x69, 0x6e, 0x74, 0x20, 0x6f, 0x63, 0x63, 0x61,
424	0x65, 0x63, 0x61, 0x74, 0x20, 0x63, 0x75, 0x70,
425	0x69, 0x64, 0x61, 0x74, 0x61, 0x74, 0x20, 0x6e,
426	0x6f, 0x6e, 0x20, 0x70, 0x72, 0x6f, 0x69, 0x64,
427	0x65, 0x6e, 0x74, 0x2c, 0x20, 0x73, 0x75, 0x6e,
428	0x74, 0x20, 0x69, 0x6e, 0x20, 0x63, 0x75, 0x6c,
429	0x70, 0x61, 0x20, 0x71, 0x75, 0x69, 0x20, 0x6f,
430	0x66, 0x66, 0x69, 0x63, 0x69, 0x61, 0x20, 0x64,
431	0x65, 0x73, 0x65, 0x72, 0x75, 0x6e, 0x74, 0x20,
432	0x6d, 0x6f, 0x6c, 0x6c, 0x69, 0x74, 0x20, 0x61,
433	0x6e, 0x69, 0x6d, 0x20, 0x69, 0x64, 0x20, 0x65,
434	0x73, 0x74, 0x20, 0x6c, 0x61, 0x62, 0x6f, 0x72,
435	0x75, 0x6d, 0x2e, 0xa, 0x4c, 0x6f, 0x72, 0x65,

436	0x6d, 0x20, 0x69, 0x70, 0x73, 0x75, 0x6d, 0x20,
437	0x64, 0x6f, 0x6c, 0x6f, 0x72, 0x20, 0x73, 0x69,
438	0x74, 0x20, 0x61, 0x6d, 0x65, 0x74, 0x2c, 0x20,
439	0x63, 0x6f, 0x6e, 0x73, 0x65, 0x63, 0x74, 0x65,
440	0x74, 0x75, 0x72, 0x20, 0x61, 0x64, 0x69, 0x70,
441	0x69, 0x73, 0x69, 0x63, 0x69, 0x6e, 0x67, 0x20,
442	0x65, 0x6c, 0x69, 0x74, 0x2c, 0x20, 0x73, 0x65,
443	0x64, 0x20, 0x64, 0x6f, 0x20, 0x65, 0x69, 0x75,
444	0x73, 0x6d, 0x6f, 0x64, 0x20, 0x74, 0x65, 0x6d,
445	0x70, 0x6f, 0x72, 0x20, 0x69, 0x6e, 0x63, 0x69,
446	0x64, 0x69, 0x64, 0x75, 0x6e, 0x74, 0x20, 0x75,
447	0x74, 0x20, 0x6c, 0x61, 0x62, 0x6f, 0x72, 0x65,
448	0x20, 0x65, 0x74, 0x20, 0x64, 0x6f, 0x6c, 0x6f,
449	0x72, 0x65, 0x20, 0x6d, 0x61, 0x67, 0x6e, 0x61,
450	0x20, 0x61, 0x6c, 0x69, 0x71, 0x75, 0x61, 0x2e,
451	0x20, 0x55, 0x74, 0x20, 0x65, 0x6e, 0x69, 0x6d,
452	0x20, 0x61, 0x64, 0x20, 0x6d, 0x69, 0x6e, 0x69,
453	0x6d, 0x20, 0x76, 0x65, 0x6e, 0x69, 0x61, 0x6d,
454	0x2c, 0x20, 0x71, 0x75, 0x69, 0x73, 0x20, 0x6e,
455	0x6f, 0x73, 0x74, 0x72, 0x75, 0x64, 0x20, 0x65,
456	0x78, 0x65, 0x72, 0x63, 0x69, 0x74, 0x61, 0x74,
457	0x69, 0x6f, 0x6e, 0x20, 0x75, 0x6c, 0x6c, 0x61,
458	0x6d, 0x63, 0x6f, 0x20, 0x6c, 0x61, 0x62, 0x6f,
459	0x72, 0x69, 0x73, 0x20, 0x6e, 0x69, 0x73, 0x69,
460	0x20, 0x75, 0x74, 0x20, 0x61, 0x6c, 0x69, 0x71,
461	0x75, 0x69, 0x70, 0x20, 0x65, 0x78, 0x20, 0x65,
462	0x61, 0x20, 0x63, 0x6f, 0x6d, 0x6d, 0x6f, 0x64,
463	0x6f, 0x20, 0x63, 0x6f, 0x6e, 0x73, 0x65, 0x71,
464	0x75, 0x61, 0x74, 0x2e, 0x20, 0x44, 0x75, 0x69,
465	0x73, 0x20, 0x61, 0x75, 0x74, 0x65, 0x20, 0x69,
466	0x72, 0x75, 0x72, 0x65, 0x20, 0x64, 0x6f, 0x6c,
467	0x6f, 0x72, 0x20, 0x69, 0x6e, 0x20, 0x72, 0x65,
468	0x70, 0x72, 0x65, 0x68, 0x65, 0x6e, 0x64, 0x65,
469	0x72, 0x69, 0x74, 0x20, 0x69, 0x6e, 0x20, 0x76,
470	0x6f, 0x6c, 0x75, 0x70, 0x74, 0x61, 0x74, 0x65,
471	0x20, 0x76, 0x65, 0x6c, 0x69, 0x74, 0x20, 0x65,
472	0x73, 0x73, 0x65, 0x20, 0x63, 0x69, 0x6c, 0x6c,
473	0x75, 0x6d, 0x20, 0x64, 0x6f, 0x6c, 0x6f, 0x72,
474	0x65, 0x20, 0x65, 0x75, 0x20, 0x66, 0x75, 0x67,
475	0x69, 0x61, 0x74, 0x20, 0x6e, 0x75, 0x6c, 0x6c,
476	0x61, 0x20, 0x70, 0x61, 0x72, 0x69, 0x61, 0x74,
477	0x75, 0x72, 0x2e, 0x20, 0x45, 0x78, 0x63, 0x65,
478	0x70, 0x74, 0x65, 0x75, 0x72, 0x20, 0x73, 0x69,
479	0x6e, 0x74, 0x20, 0x6f, 0x63, 0x63, 0x61, 0x65,
480	0x63, 0x61, 0x74, 0x20, 0x63, 0x75, 0x70, 0x69,
481	0x64, 0x61, 0x74, 0x61, 0x74, 0x20, 0x6e, 0x6f,
482	0x6e, 0x20, 0x70, 0x72, 0x6f, 0x69, 0x64, 0x65,
483	0x6e, 0x74, 0x2c, 0x20, 0x73, 0x75, 0x6e, 0x74,

484 0x20, 0x69, 0x6e, 0x20, 0x63, 0x75, 0x6c, 0x70,
485 0x61, 0x20, 0x71, 0x75, 0x69, 0x20, 0x6f, 0x66,
486 0x66, 0x69, 0x63, 0x69, 0x61, 0x20, 0x64, 0x65,
487 0x73, 0x65, 0x72, 0x75, 0x6e, 0x74, 0x20, 0x6d,
488 0x6f, 0x6c, 0x6c, 0x69, 0x74, 0x20, 0x61, 0x6e,
489 0x69, 0x6d, 0x20, 0x69, 0x64, 0x20, 0x65, 0x73,
490 0x74, 0x20, 0x6c, 0x61, 0x62, 0x6f, 0x72, 0x75,
491 0x6d, 0x2e, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
492 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
493 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
494 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
495 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
496 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
497 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
498 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
499 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
500 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
501 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
502 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
503 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
504 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
505 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
506 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
507 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
508 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
509 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
510 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
511 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
512 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
513 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
514 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
515 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
516 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
517 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
518 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
519 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
520 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
521 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
522 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
523 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
524 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
525 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
526 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
527 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
528 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
529 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
530 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
531 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0,

532];

VI.XIII /tools/MicroFS/Cargo.toml

```
1 [package]
2 name = "micro_fs"
3 version = "0.1.0"
4 authors = ["orpheeantoniadis <orphee.antoniadis@gmail.com>"]
5
6 [dependencies]
7 clap = {version = "~2.31", features = ["yaml"]}
```


VII /user

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

```
1  #![no_std]
2
3  extern crate ulibc;
4  use ulibc::*;
5  use io::*;
6  use mem::*;
7
8  #[no_mangle]
9  pub extern fn main() {
10     clear();
11     puts("Starting demo.\n");
12
13     println!("\nIO demo :\n");
14     println!("Executing hello app..");
15     exec("hello");
16     println!("Waiting on keypressed..");
17     while keypressed() == 0 {}
18     getc();
19     println!("Waiting 1 sec..");
20     sleep(1000);
21     println!("Waiting on getc..");
22     let key = getc();
23     println!("{}", pressed., key as u8 as char);
24     sleep(3000);
25     clear();
26
27     println!("File system demo :\n");
28     println!("Opening file splash.txt..");
29     let fd = file_open("splash.txt");
30     if fd != -1 {
31         println!("Reading file splash.txt..");
32         let mut data = [0;MAX_STR_LEN];
33         file_read(fd as u32, &mut data[0], MAX_STR_LEN as u32);
34         println!("{}", bytes_to_str(&data));
35         println!("Closing file splash.txt..");
36         file_close(fd as u32);
37     }
38     println!("Iterating all the files..", );
39     let it = file_iterator();
40     let mut bytes = [0;MAX_FILENAME_LENGTH];
41     while file_next(&bytes[0], &it) != -1 {
42         {
43             let filename = bytes_to_str(&bytes);
44             println!("{}", {}, filename, file_stat(filename).size);
45         }
46     }
```

```

46     bytes = [0;MAX_FILENAME_LENGTH];
47 }
48 sleep(3000);
49 clear();
50
51 println!("Memory management demo :\n");
52 unsafe {
53     println!("Allocating 1M on the heap..", );
54     let addr1 = malloc(0x100000);
55     *(addr1 as *mut u8) = 42;
56     println!("addr1 = 0x{:x}, [addr1] = 0x{:x}", addr1, *(addr1 as
57 *mut u8));
58     println!("Allocating 256 bytes on the heap..");
59     let addr2 = malloc(0x100);
60     *(addr2 as *mut u8) = 0x42;
61     println!("addr2 = 0x{:x}, [addr2] = 0x{:x}", addr2, *(addr2 as
62 *mut u8));
63     println!("Freeing addr1..");
64     free(addr1);
65     println!("Allocating 4KB on the heap..");
66     let addr3 = malloc(0x1000);
67     *(addr3 as *mut u8) = 12;
68     println!("addr3 = 0x{:x}, [addr3] = 0x{:x}", addr3, *(addr3 as
69 *mut u8));
70     println!("Alloc list state :");
71     print_kmalloc_list();
72     println!("Freeing addr2..");
73     free(addr2);
74     println!("Freeing addr3..");
75     free(addr3);
76     println!("Alloc list state :");
77     print_kmalloc_list();
78 }
79 println!("Ok.");
80 }

```

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

```

1  #![no_std]
2
3  extern crate ulibc;
4  use ulibc::*;
5  use io::*;
6
7  #[no_mangle]
8  pub extern fn main() {
9      puts("Hello world!\n");

```

10 }

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

```
1  #![no_std]
2
3  extern crate ulibc;
4  use ulibc::*;
5  use io::*;
6  use core::str::FromStr;
7
8  const MAX_CMD_LEN: usize = MAX_FILENAME_LENGTH;
9
10 fn cat(filename: &str) {
11     let fd = file_open(filename);
12     if fd != -1 {
13         let mut data = [0; MAX_STR_LEN];
14         while file_read(fd as u32, &mut data[0], MAX_STR_LEN as u32) != 0
15     {
16         {
17             let content = bytes_to_str(&data);
18             if content == "\0" {
19                 println!("cat: {}: Not a text file", filename);
20                 break;
21             } else {
22                 println!("{}", content);
23             }
24             data = [0; MAX_STR_LEN];
25         }
26         file_close(fd as u32);
27     } else {
28         println!("cat: {}: No such file or directory", filename);
29     }
30 }
31
32 fn help() {
33     puts("\n");
34     puts("ls          : list files present in the file system\n");
35     puts("cat <file>   : dump the content of <file> to the screen\n");
36     puts("clear        : clear the screen\n");
37     puts("<prog>       : execute the program <prog>.\n");
38     puts("sleep <ms>   : sleep the specified number of milliseconds\n");
39     puts("exit         : exit the shell\n");
40 }
41
42 fn ls() {
43     let it = file_iterator();
```

```

44     let mut bytes = [0;MAX_FILENAME_LENGTH];
45     while file_next(&bytes[0], &it) != -1 {
46         {
47             let filename = bytes_to_str(&bytes);
48             println!("{}", filename, file_stat(filename).size);
49         }
50         bytes = [0;MAX_FILENAME_LENGTH];
51     }
52 }
53
54 fn read_cmd(cmd: *mut u8) {
55     let mut key = 0;
56     let mut cnt = 0;
57     while key != b'\n' {
58         if key == 0x8 {
59             if cnt > 0 {
60                 cnt -= 1;
61                 unsafe { *cmd.offset(cnt as isize) = 0; }
62                 putc(key);
63             }
64             } else if key != b'\t' && key != b'\0' {
65                 unsafe { *cmd.offset(cnt as isize) = key; }
66                 cnt += 1;
67                 putc(key);
68             }
69             key = getc();
70         }
71     }
72
73     #[no_mangle]
74     pub extern fn main() {
75         loop {
76             let mut cmd : [u8;MAX_CMD_LEN] = [0;MAX_CMD_LEN];
77             print!("$ ");
78             read_cmd(&mut cmd[0]);
79             println!();
80             let mut s = String::new(bytes_to_str(&cmd));
81             let mut args = s.to_string().split_whitespace();
82             match args.next() {
83                 Some(cmd) => {
84                     let arg = match args.next() {
85                         Some(arg) => arg,
86                         _ => ""
87                     };
88                     match cmd {
89                         "cat" => cat(arg),
90                         "clear" => clear(),
91                         "exit" => break,

```

```

92         "help"  => help(),
93         "ls"    => ls(),
94         "sleep" => {
95             let ms = match u32::from_str(arg) {
96                 Ok(num) => num,
97                 Err(_) => {
98                     println!("sleep: invalid time interval
99 '{}'", arg);
100                     continue;
101                 }
102             };
103             println!("Sleeping for {}ms..", ms);
104             sleep(ms);
105         }
106     _ => { exec(cmd); }
107 }
108 _ => continue
109 }
110 }
111 println!("exit");
112 }

```

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

```

1  #![no_std]
2
3  extern crate ulibc;
4  use ulibc::*;
5  use io::*;
6
7  #![no_mangle]
8  pub extern fn main() {
9      println!("Hello world!");
10 }

```

VII.V /user/skeleton/Cargo.toml

```

1  [package]
2  name = "skeleton"
3  version = "0.1.0"
4  authors = ["orpheeantoniadis <orphee.antoniadis@gmail.com>"]
5
6  [lib]
7  name = "skeleton"
8  path = "src/skeleton.rs"
9  crate-type = ["staticlib"]

```

```

10
11 [dependencies]
12 ulibc = { path = "../ulibc" }
13
14 [profile.release]
15 lto = true
16 panic = 'abort'

```

VII.VI /user/skeleton/Makefile

```

1 ARCH = i386
2 TARGET = $(ARCH)-app
3
4 CC = xargo
5 RFLAGS = "-C relocation-model=static -C opt-level=3"
6 XFLAGS = --release -vv --target $(TARGET)
7
8 .PHONY : all build test clean
9
10 all : build
11
12 build :
13     RUST_TARGET_PATH=$(shell pwd)/.. RUSTFLAGS=$(RFLAGS) $(CC) build
14     $(XFLAGS)
15
16 clean :
17     $(CC) clean

```

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

```

1 #![no_std]
2
3 extern crate ulibc;
4 use ulibc::*;
5 use io::*;
6
7 #![no_mangle]
8 pub extern fn main() {
9     cursor_disable(true);
10    clear();
11    set_cursor(22,10);
12    let fd = file_open("splash.txt") as u32;
13    let mut data = [0;MAX_STR_LEN];
14    file_read(fd, &mut data[0], MAX_STR_LEN as u32);
15    for byte in bytes_to_str(&data).bytes() {
16        putc(byte);
17        if byte == b'\n' {

```

```

18         let cursor = (0,0);
19         get_cursor(&cursor.0, &cursor.1);
20         set_cursor(22,cursor.1);
21     }
22 }
23 file_close(fd);
24 sleep(5000);
25 clear();
26 cursor_disable(false);
27 }

```

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

```

1 use io::*;
2
3 static mut SCR: FrameBuffer =
4     [[Character::null();BUFFER_WIDTH];BUFFER_HEIGHT];
5
6 pub struct Image {
7     content: *const u8,
8     width: usize,
9     height: usize
10 }
11
12 pub struct Window {
13     img: Image,
14     x: usize,
15     y: usize,
16     color: ColorAttribute
17 }
18
19 pub fn init_scr() {
20     unsafe {
21         copy_scr(&SCR);
22         set_cursor(0, 0);
23         cursor_disable(true);
24     }
25 }
26
27 pub fn destroy_scr() {
28     unsafe {
29         SCR = [[Character::null();BUFFER_WIDTH];BUFFER_HEIGHT];
30         copy_scr(&SCR);
31         set_cursor(0, 0);
32         cursor_disable(false);
33     }
34 }

```

```

35 pub fn update_scr() {
36     unsafe {
37         copy_scr(&SCR);
38     }
39 }
40
41 pub fn display_window(w: Window) {
42     unsafe {
43         for i in 0..w.img.height {
44             for j in 0..w.img.width {
45                 let ascii = *w.img.content.offset((j + i * w.img.width)
46 as isize);
47                 SCR[i+w.x][j+w.y] = Character::new(ascii, w.color);
48             }
49         }
50         update_scr();
51     }
52 }
53 impl Image {
54     pub fn new(data: &str, width: usize, height: usize) -> Image {
55         Image {
56             content: data.as_ptr(),
57             width: width,
58             height: height
59         }
60     }
61 }
62
63 impl Window {
64     pub fn new(img: Image, x: usize, y: usize, background: Color,
65 foreground: Color) -> Window {
66         Window {
67             img: img,
68             x: x,
69             y: y,
70             color: ColorAttribute::new(background, foreground)
71         }
72     }
73 }

```

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

```

1  #![macro_use]
2
3  use core::fmt::{Error, Write, Arguments};
4  pub use common::*;
5

```



```

6  extern "C" {
7      pub fn syscall(nb: Syscall, arg1: u32, arg2: u32, arg3: u32, arg4:
u32) -> i32;
8  }
9
10 pub struct Stdout {}
11
12 impl Write for Stdout {
13     fn write_str(&mut self, s: &str) -> Result<(), Error> {
14         unsafe { syscall(Syscall::Puts, &String::new(s) as *const String
as u32, 0, 0, 0); }
15         Ok(())
16     }
17 }
18
19 pub fn write_fmt(args: Arguments) {
20     Stdout {}.write_fmt(args).ok();
21 }
22
23 #[macro_export]
24 macro_rules! print {
25     ($($arg:tt)*) => (write_fmt(format_args!($($arg)*)));
26 }
27
28 #[macro_export]
29 macro_rules! println {
30     () => (print!("\n"));
31     ($fmt:expr) => (print!(concat!($fmt, "\n")));
32     ($fmt:expr, $($arg:tt)*) => (print!(concat!($fmt, "\n"), $($arg)*));
33 }
34
35 pub fn clear() {
36     copy_scr(&[[Character::null();BUFFER_WIDTH];BUFFER_HEIGHT]);
37     set_cursor(0, 0);
38 }
39
40 pub fn puts(s: &str) {
41     unsafe {
42         syscall(Syscall::Puts, String::new(s).as_ptr() as u32, 0, 0, 0);
43     }
44 }
45
46 pub fn putc(byte: u8) {
47     unsafe {
48         syscall(Syscall::Putc, byte as u32, 0, 0, 0);
49     }
50 }
51

```

```

52 pub fn exec(s: &str) -> i32 {
53     unsafe {
54         syscall(Syscall::Exec, String::new(s).as_ptr() as u32, 0, 0, 0)
55     }
56 }
57
58 pub fn keypressed() -> i32 {
59     unsafe {
60         syscall(Syscall::Keypressed, 0, 0, 0, 0)
61     }
62 }
63
64 pub fn getc() -> u8 {
65     unsafe {
66         syscall(Syscall::Getc, 0, 0, 0, 0) as u8
67     }
68 }
69
70 pub fn file_stat(s: &str) -> Stat {
71     let mut stat = Stat::null();
72     unsafe {
73         syscall(Syscall::FileStat, String::new(s).as_ptr() as u32,
74 stat.as_ptr() as u32, 0, 0);
75     }
76     return stat;
77 }
78
79 pub fn file_open(s: &str) -> i32 {
80     unsafe {
81         syscall(Syscall::FileOpen, String::new(s).as_ptr() as u32, 0, 0,
82 0)
83     }
84 }
85
86 pub fn file_close(fd: u32) -> i32 {
87     unsafe {
88         syscall(Syscall::FileClose, fd, 0, 0, 0)
89     }
90 }
91
92 pub fn file_read(fd: u32, buf: *mut u8, n: u32) -> i32 {
93     unsafe {
94         syscall(Syscall::FileRead, fd, buf as u32, n, 0)
95     }
96 }
97
98 pub fn file_seek(fd: u32, offset: u32) -> i32 {
99     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
122 pub fn sleep(ms: u32) {
123     unsafe {
124         syscall(Syscall::Sleep, ms, 0, 0, 0);
125     }
126 }
127
128 pub fn set_cursor(x: u32, y: u32) {
129     unsafe {
130         syscall(Syscall::SetCursor, x, y, 0, 0);
131     }
132 }
133
134 pub fn get_cursor(x: *const u32, y: *const u32) {
135     unsafe {
136         syscall(Syscall::GetCursor, x as u32, y as u32, 0, 0);
137     }
138 }
139
140 pub fn cursor_disable(cd: bool) {
141     unsafe {
142         if cd {
143             syscall(Syscall::CursorDisable, 1, 0, 0, 0);
144         } else {
145             syscall(Syscall::CursorDisable, 0, 0, 0, 0);

```

```

146     }
147 }
148 }
149
150 pub fn copy_scr(scr: *const FrameBuffer) {
151     unsafe {
152         syscall(Syscall::CopyScr, scr as u32, 0, 0, 0);
153     }
154 }

```

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

```

1  // extern crate alloc;
2  use core::mem::size_of;
3  use rlibc::{memset, memcpy};
4  use io::*;
5
6  const FRAME_SIZE: usize = 0x1000;
7  const HEAP_END: u32 = 0xffffffff;
8
9  static mut HEAP_START: u32 = 0;
10 static mut HEAP_SIZE: usize = 0;
11
12 #[derive(Debug, Clone, Copy)]
13 #[repr(C, align(16))]
14 struct Header {
15     previous: u32,
16     next: u32,
17     size: usize,
18     free: bool
19 }
20
21 macro_rules! align {
22     ($size:expr) => {
23         if ($size & 0xffffffff0) != $size {
24             ($size & 0xffffffff0) + 0x10
25         } else {
26             $size
27         };
28     }
29 }
30
31 fn heap_init() {
32     unsafe {
33         if HEAP_START == 0 {
34             HEAP_START = syscall(Syscall::AllocFrame, 0, 0, 0, 0) as u32;
35             HEAP_SIZE = (HEAP_END - HEAP_START) as usize;
36             memset(HEAP_START as *mut u8, 0, FRAME_SIZE);

```

```

37         memcpy(HEAP_START as *mut u8, Header::null(0,
HEAP_SIZE).as_ptr(), size_of::<Header>());
38     }
39 }
40 }
41
42 pub fn malloc(size: usize) -> u32 {
43     heap_init();
44     let aligned_size = align!(size);
45     let mut addr = empty_block(aligned_size);
46     let mut block = Header::from_ptr(addr as *mut u8);
47     if block.size >= aligned_size && block.free {
48         if block.next == 0 {
49             block.insert_tail(addr, aligned_size);
50         } else {
51             block.insert(addr, aligned_size);
52         }
53         addr += size_of::<Header>() as u32;
54         unsafe { memset(addr as *mut u8, 0, aligned_size); }
55         return addr;
56     }
57     return 0;
58 }
59
60 pub fn free(addr: u32) {
61     unsafe {
62         let mut header_addr = addr - size_of::<Header>() as u32;
63         let mut header = Header::from_ptr(header_addr as *const u8);
64         if !header.free {
65             let start = header_addr;
66             let mut end = header.next;
67             if header.previous != 0 {
68                 let previous = Header::from_ptr(header.previous as *const
u8);
69                 if previous.free {
70                     header_addr = header.previous;
71                     header.previous = previous.previous;
72                     header.size += previous.size + size_of::<Header>();
73                 }
74             }
75             if header.next != 0 {
76                 let mut next = Header::from_ptr(header.next as *const u8);
77                 if next.free {
78                     header.next = next.next;
79                     header.size += next.size + size_of::<Header>();
80                 }
81                 if header.next != 0 {
82                     end = header.next;

```

```

83         next = Header::from_ptr(header.next as *const u8);
84         if next.previous != header_addr {
85             next.previous = header_addr;
86             memcpy(header.next as *mut u8, next.as_ptr(),
size_of::<Header>());
87         }
88     }
89 }
90 header.free = true;
91 memcpy(header_addr as *mut u8, header.as_ptr(),
size_of::<Header>());
92     // free unused page tables
93     let start_idx = start as usize / FRAME_SIZE;
94     let mut end_idx = end as usize / FRAME_SIZE;
95     if header.next == 0 {
96         end_idx += 1;
97     }
98     for i in start_idx..end_idx {
99         syscall(Syscall::FreeFrame, (i * FRAME_SIZE) as u32, 0,
0, 0);
100     }
101 }
102 }
103 }
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,
120 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(),
120 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 -
120 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(),
120 size_of::<Header>());
161                 memcpy(self.next as *mut u8, next_header.as_ptr(),
120 size_of::<Header>());
162                 memcpy(tmp as *mut u8, tmp_header.as_ptr(),
120 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
191 fn from_ptr(ptr: *const u8) -> Header {
192     unsafe {
193         *(ptr as *const Header)
194     }
195 }
196 }

```

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

```

1  #![feature(lang_items)]
2  // #![feature(alloc, global_allocator, allocator_api)]
3  #![no_std]
4
5  extern crate common;
6  pub use common::*;
7
8  extern crate rlibc;
9  pub use rlibc::*;
10
11 pub mod io;
12 pub mod curses;
13 pub mod mem;
14
15 #[lang = "panic_fmt"]

```



```

16  #[no_mangle]
17  pub extern "C" fn panic_fmt() -> ! {
18      loop{}
19  }
20
21  #[no_mangle]
22  pub extern "C" fn __floatundisf() {
23      loop {}
24  }

```

VII.XII /user/ulibc/Cargo.toml

```

1  [package]
2  name = "ulibc"
3  version = "0.1.0"
4  authors = ["orpheeantoniadis <orphee.antoniadis@gmail.com>"]
5
6  [lib]
7  name = "ulibc"
8  path = "src/ulibc.rs"
9
10 [dependencies]
11 rlibc = "1.0"
12 common = { path = "../common" }

```

VII.XIII /user/app.ld

```

1  OUTPUT_FORMAT("binary")
2
3  SECTIONS {
4      . = 0x0;                /* first section located at 0 */
5
6      .entrypoint ALIGN(4):    /* entry point: must be located at 0 */
7      {
8          *(.entrypoint)
9      }
10
11     .text ALIGN(4) :         /* code */
12     {
13         *(.text*)
14     }
15
16     .rodata ALIGN(4) :       /* read-only data */
17     {
18         *(.rodata*)
19     }
20

```

```

21     .data ALIGN(4) :          /* initialized data */
22     {
23         *(.data*)
24     }
25
26     .bss ALIGN(4) :           /* uninitialized data */
27     {
28         *(COMMON)
29         *(.bss*)
30     }
31 }

```

VII.XIV /user/entrypoint_asm.s

```

1  extern main
2
3  section .entrypoint
4  align 4
5
6      mov     [stack_ptr],esp
7      call    main
8      mov     esp,[stack_ptr]
9      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,
7  uint32_t arg4);
8  syscall:
9      ; parameters cannot be passed into the stack because the
10     trap/interrupt gate
11     ; performs a stack switch (from user stack to kernel stack). By the
12     time we're
13     ; in the syscall handler we're accessing the kernel stack
14     (tss.ss/tss.esp).
15     push    ebp
16     mov     esp,ebp
17
18     ; save all general registers since we modify them below
19     ; eax is not saved as it's used to store the syscall's return value
20     push    ebx
21     push    ecx
22     push    edx
23     push    esi
24     push    edi
25
26     mov     eax,[ebp+8]
27     mov     ebx,[ebp+12]
28     mov     ecx,[ebp+16]
29     mov     edx,[ebp+20]
30     mov     esi,[ebp+24]
31     int     48
32
33     ; restore all general registers
34     pop     edi
35     pop     esi
36     pop     edx
37     pop     ecx
38     pop     ebx
39
40     mov     esp,ebp
41     pop     ebp
42     ret

```

VIII /Makefile

```
1 KERNEL_PATH = kernel/
2
3 .PHONY: all build run test doc clean mrproper
4
5 all: build
6
7 build:
8     $(MAKE) -C $(KERNEL_PATH)
9
10 run:
11     $(MAKE) -C $(KERNEL_PATH) run
12
13 test:
14     $(MAKE) -C $(KERNEL_PATH) test
15
16 doc:
17     $(MAKE) -C $(KERNEL_PATH) doc
18
19 clean:
20     $(MAKE) -C $(KERNEL_PATH) clean
21
22 mrproper:
23     $(MAKE) -C $(KERNEL_PATH) mrproper
```