# The K Project

## Introduction

LSE Team

EPITA

May 06, 2019

## Introduction

Figure: K running ‘skate’

## Assignment

- Serial driver
- Segmentation
- Events
- Keyboard
- Timer
- ATAPI driver
- File system
- Binary loading
- Syscalls
- VGA driver
- Bonus: Sound driver, Console driver, ...

## Intermediate handout

- Basic serial driver
- Segmentation initialization

## What is a kernel ?

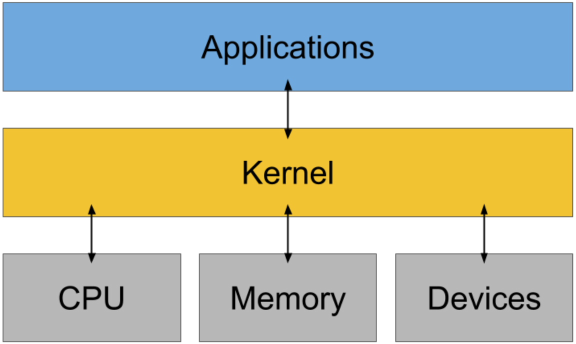

Figure: Operating system layout

## Git Repository

```
git clone https://github.com/lse/k.git
```

## Planning

- First Part
  - Start: March, 07th
  - Deadline: March, 31th
- Second Part
  - Start: April, 22th
  - Deadline: July, 07th

## Source Tree

```
.
|-- k
|   |-- compiler.h
|   |-- crt0.S
|   |-- elf.h
|   |-- include
|   |-- io.h
|   |-- k.c
|   |-- k.lds
|   |-- libvga.c
|   |-- libvga.h
|   +-- multiboot.h
|-- libs
|   |-- libc
|   +-- libk
|-- roms/
+-- tools/
```

## First step

```
#include "multiboot.h"
#include "kstd.h"

void    k_main(unsigned long          magic,
               multiboot_info_t*      info)
{
        (void) magic;
        (void) info;

        char star[4] = "|/-\\";
        char *fb = (void *)0xb8000;

        for (unsigned i = 0; ; )
                *fb = star[i++ % 4];
}
```

## Documentation

- `http://k.lse.epita.fr/`
- `http://intel.com/products/processor/manuals/`

## DEBUG

### Launch your K

```
$ qemu-system-i386 -cdrom k.iso [ -enable-kvm ]
```

### Have QEMU wait for your debugger to hook:

- Add “`-s -S`” to QEMU options

### Launch gdb and hook to QEMU

```
$> gdb k/k
$(gdb)> target remote localhost:1234
```

### Set your breakpoint and continue

Don’t forget to build with debug options !

## x86 Registers:

- General purpose registers
- Segment registers
- Flags
- Control & Memory registers
- Tons of others (XMM0-7...)

## General purpose registers

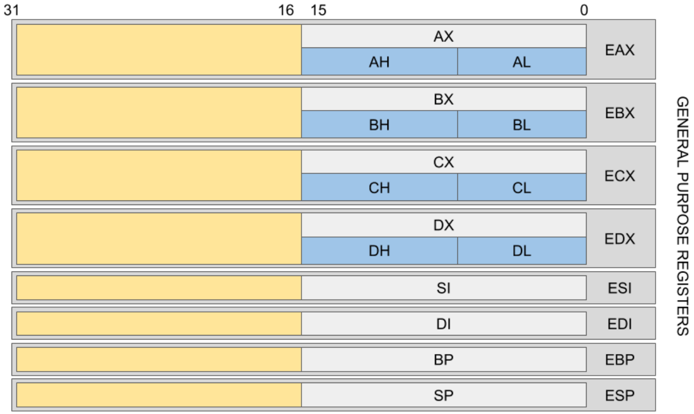

Figure: General purpose registers layout

# IP and FLAGS registers

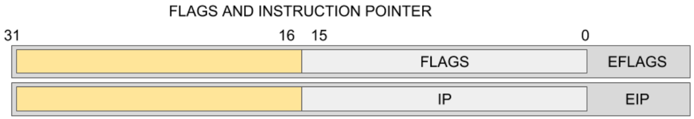

Figure: EIP/IP and EFLAGS/FLAGS

# Flags register

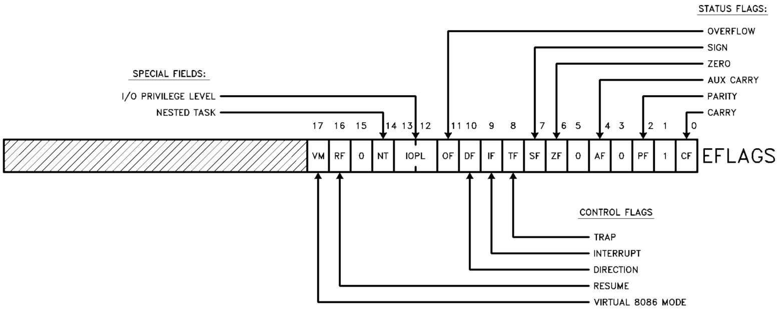

Figure: Flags layout

# Rings

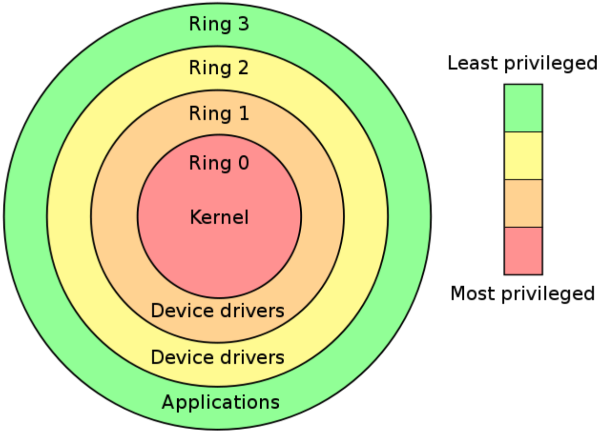

Figure: x86 privileges rings

# Calling Conventions

**C declaration:**

```
pushl %eax ; arg3
pushl %ebx ; arg2
pushl %ecx ; arg1
call  foo  ; foo(arg1, arg2, arg3)
```

**Think of call as:**

```
pushl %eip
%eip = ADDRESS
```

**Think of ret as:**

```
popl %eip
```

## Exemple

**Asm exemple for sum(int, int)**

```
sum:
    pushl %ebp
    movl %esp, %ebp
    movl 8(%ebp), %eax ; put first arg in %eax
    addl 12(%ebp), %eax ; add second arg to %eax
    movl %ebp, %esp
    popl %ebp
    ret
```

## ASM inline (GAS syntax)

**Basic syntax:**

```
__asm__("movb %ah, (%ebx)");
```

**Tell GCC/GAS not to optimize your code:**

```
asm volatile ("movl $0, %eax");
```

**Note**

You can either write GNU keywords and specifiers with or without double underscore around them to avoid name conflicts (`asm` or `__asm__`, `volatile` or `__volatile__`).

## ASM inline - next level

**ASM inline template**

```
__asm__("[your assembly code]"
: output operands /* optional */
: input operands /* optional */
: list of clobbered /* optional */
);
```

# ASM inline - next level

## Different output/input constraints:

```
http://gcc.gnu.org/onlinedocs/gcc/Constraints.html

"m" : memory operand
"r" : register operand
```

## Constraint modifiers

```
https://gcc.gnu.org/onlinedocs/gcc/Modifiers.html#
Modifiers

- "=" : Write Only
- "+" : Read/Write
```

## Different clobbers

```
memory
[register names]
```

# ASM Inline - example

```
asm volatile("outb %0, %1\n\t"
             : /* No output */
             : "a" (val), "d" (port));

asm volatile("inb %1, %0\n\t"
             : "=&a" (res)
             : "d" (port));
```

# Intel vs AT&T

| Intel Code | AT&T Code |
|---|---|
| `mov eax,1` | `mov $1,%eax` |
| `int 80h` | `int $0x80` |
| `mov ebx,eax` | `mov %eax,%ebx` |
| `mov eax,[ebx+3]` | `mov 3(%ebx),%eax` |
| `mov eax,[ebx+20h]` | `mov 0x20(%ebx),%eax` |
| `lea eax,[ebx+ecx]` | `lea (%ebx, %ecx),%eax` |

## Bitfields

```c
struct {
    unsigned char field_a : 1; // max value is 0b1
    unsigned char field_b : 2; // max value is 0b11
    unsigned char field_c : 5; // max value is 0x1F
} bitfields;
```

**Note**

`sizeof(bitfields)` is equal to `sizeof(unsigned char)`.

## Packed Structs

### Not Packed

```
struct {
    unsigned char   a; // aligns with 3 bytes
    unsigned int    b; // aligned
    unsigned char   c; // aligns with 3 bytes
} foo;
```

### Note

`sizeof(foo)` gives 12 (1 + 3 padding + 4 + 1 + 3 padding).

# Packed Structs

## Packed

```
struct {
    unsigned char   a;
    unsigned int    b;
    unsigned char   c;
}__attribute__((packed)) bar;
```

## Note

`sizeof(bar)` gives 6, struct is memory packed and no padding is inserted.

## Your first kernel function !

**write()**

```
int write(const char *buf, size_t count);
```

**Note**

write() sends to COM1

**Note**

printf() is available in your kernel and uses write()

## PortIO

- Separated adress space
- 2ˆ16 adresses
- in/out x86 instructions family
- Serial/PIC/PIT/Keyboard

## Serial initialization

- COM1 + 3: (8 bits length) | (No parity)
- COM1 + 2: (FIFO) | (Interrupt trigger level 14 bytes) | (Clear transmit FIFO) | (Clear receive FIFO)
- COM1 + 1: Enable Transmitter Holding Register Empty Interrupt

## Serial

- A line ends with `\r\n`
- You can redirect the serial output with:

```
qemu-system-i386 [...] -serial stdio
```

## Contact

- k[at]lse.epita.fr
- labos.lse with [K] tag
- #k (irc.rezosup.org)
- guillaume.pagnoux[at]lse.epita.fr
- tom.decrette[at]lse.epita.fr