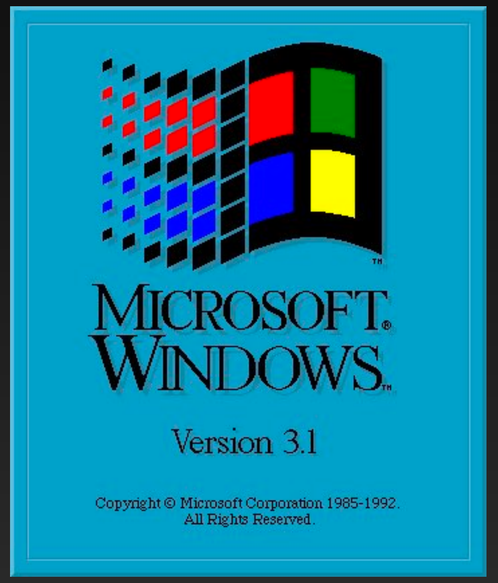
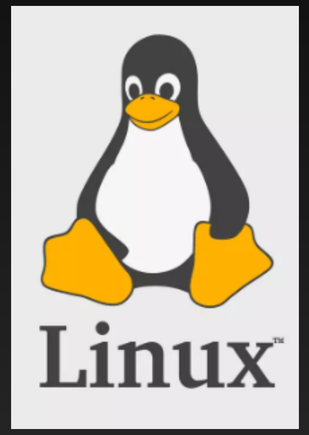
Operating Systems

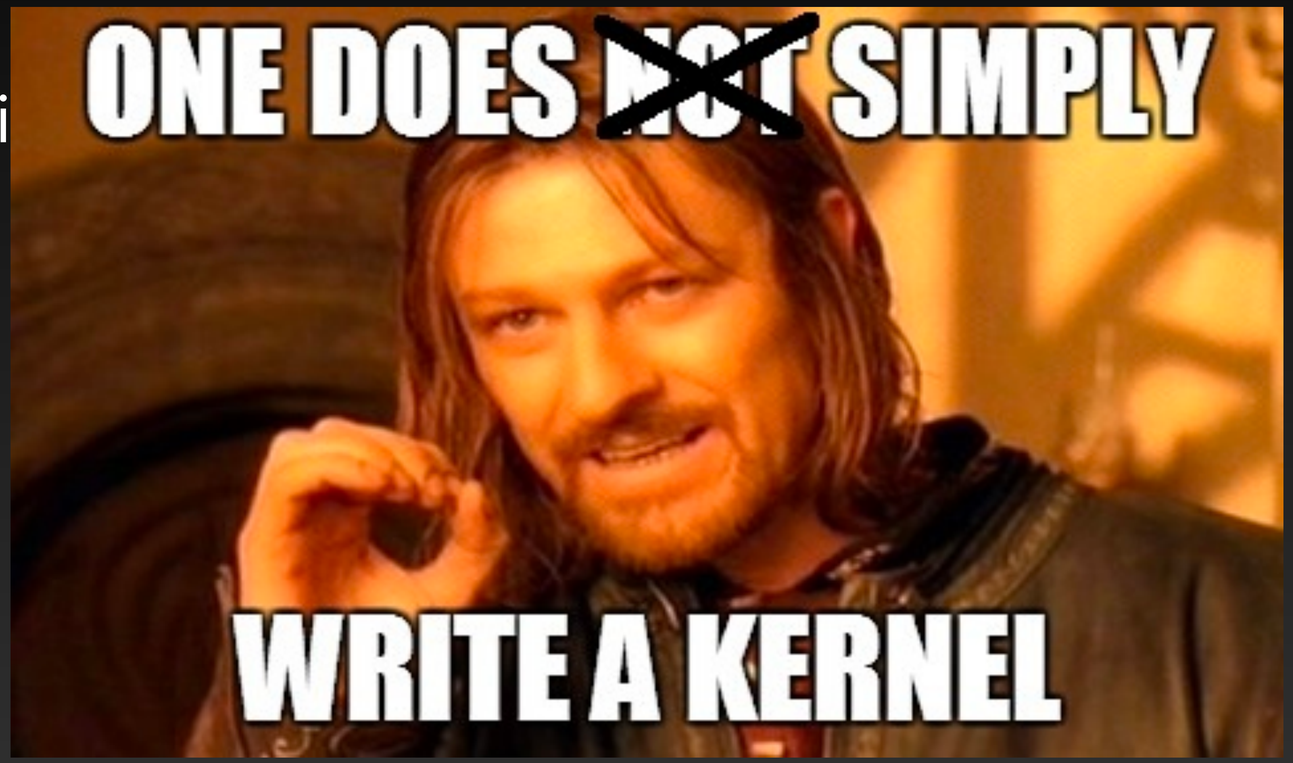
What is your favourite OS?





Did you ever want write your own OS?

- I'll try to show i



What do you need

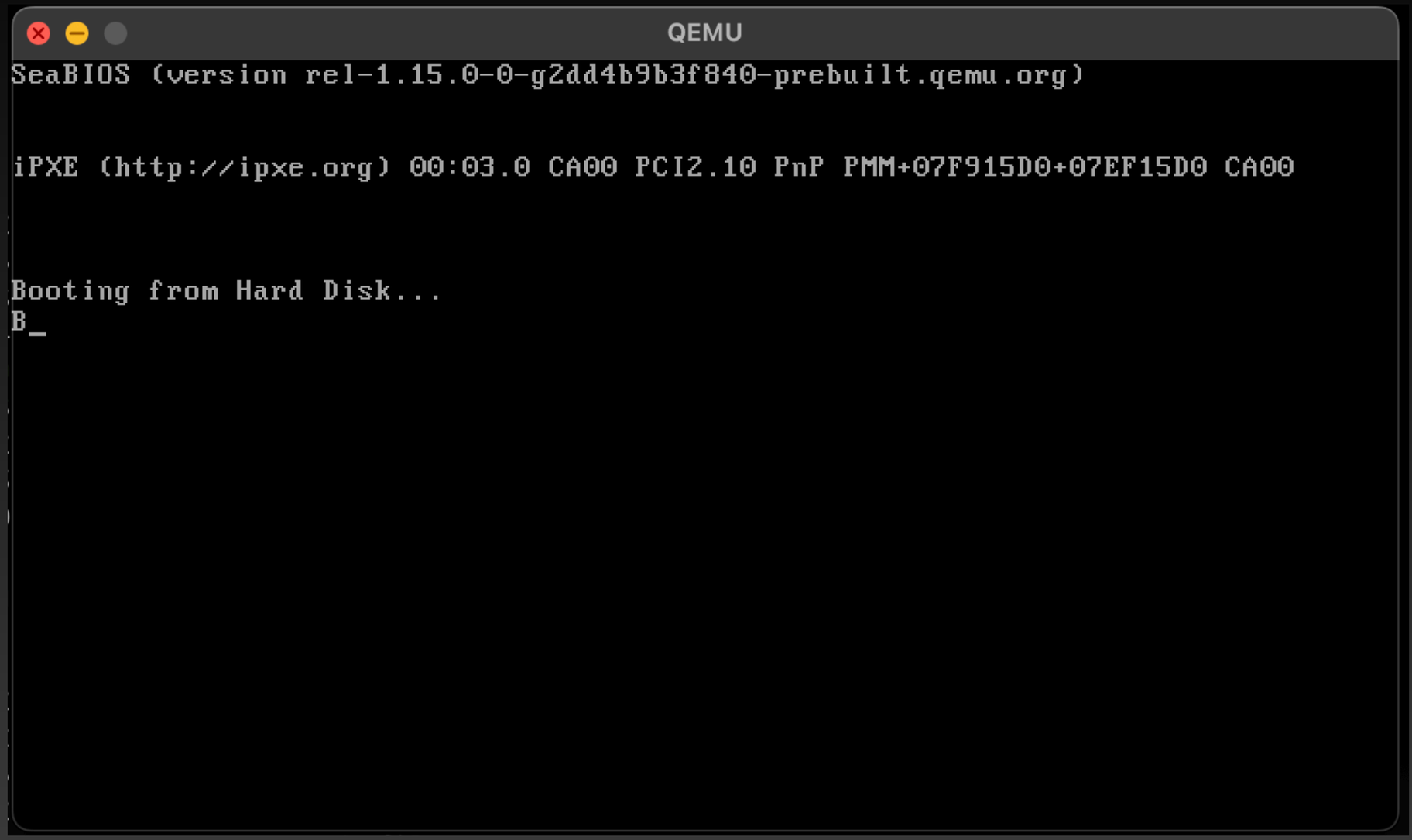
- Your favourite assembler (NASM, TASM, FASM) - this is evertything that you need for basic OS
- Cross compiler (you need compile a compiler from sources)
- Bootloader GRUB - if you want run your OS on real hardware
- Emulator Bochs/QEemu - for development
- Optional debugger (GDB for linux /LLDB on macOS)

BIOS mode

```
[org 0x7c00] ; memory addreses not starting from zero but offset 0x7c00
mov ah, 0x0e ; teletype mode

mov al, [string] ; first character(not addres) []-pointer dereference
int 0x10         ; bios function - print character from al register

string: db “Hello BIOS!”, 0

jmp $            ; jump to current address (infinite loop)
times 510-($-$$) db 0 ; align
db 0x55, 0xaa
```

```
Run commands:
nasm -g example.asm -f bin -o boot.bin
qemu-system-i386 -hda boot.bin
```

BIOS mode


QEMU
SeaBIOS (version rel-1.15.0-0-g2dd4b9b3f840-prebuilt.qemu.org)
iPXE (http://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+07F915D0+07EF15D0 CA00
Booting from Hard Disk...
B_

Bootloader

```
[org 0x7c00]


KERNEL_LOCATION equ 0x1000


mov [BOOT_DISK], dl     ; Stores the boot disk number

mov bx, OK
call PrintString

xor ax, ax          ; clear bits of ax
mov es, ax          ; set es to 0
mov ds, ax          ; set ds to 0
mov bp, 0x8000      ; stack base
mov sp, bp          ; stack pointer to stack base
                    ; A:B = A*d16 + B
mov bx, KERNEL_LOCATION ; ES:BX is the location to read from, e.g. 0x0000:0x9000 = 0x00000 + 0x9000 = 0x9000
mov dh, 50          ; read 35 sectors (blank sectors: empty_end)

call readDisk

call switchToPM
    jmp $
```

Switch to protected mode

```
[bits 16]

switchToPM:
    cli                         ; turn off interrupts, it's crucial operation
    lgdt [GDT_descriptor]       ; Load GDT

    mov eax, cr0
    or eax, 0x1           ; sets the first bit of eax to 1, while leaving the rest intact
    mov cr0, eax

    jmp CODE_SEG:init_pm ; far jump to code segment, flushes CPU cache, removing real mode instructions

[bits 32]

init_pm:
    mov ax, DATA_SEG
    mov ds, ax
    mov ss, ax
    mov es, ax
    mov fs, ax
    mov gs, ax

    mov ebp, 0x90000        ; 32 bit stack base pointer
    mov esp, ebp
    call BEGIN_PM
```

```
GDT_start:              ; to define the Global Descriptor Table we define bytes, words and double words
                        ; to write them "raw" in the bin file (like we did in the disk reading example, or to define the
                        ; end of the bootsector)

GDT_null:               ; mandatory NULL descriptor
    dd 0x0
    dd 0x0

GDT_code:               ; code segment descriptor
    dw 0xffff           ; limit, bits 0-15 (completed at line 14)
    dw 0x0              ; Base, bits 0-15
    db 0x0              ; Base, bits 16-23 (completed at line 15)
    db 0b10011010       ; Flags (look at paper for meaning)
    db 0b11001111       ; Flags, Limit bits 16-19
    db 0x0              ; Base, bits 24-31

GDT_data:               ; data segment descriptor
    dw 0xffff           ; limit, bits 0-15
    dw 0x0              ; Base, bits 0-15
    db 0x0              ; Base, bits 16-23
    db 0b10010010       ; Flags (look at paper for meaning)         DATA FLAGS ARE DIFFERENT IN DATA SEGMENT
    db 0b11001111       ; Flags, Limit bits 16-19
    db 0x0              ; Base, bits 24-31

GDT_end:

GDT_descriptor:
    dw GDT_end - GDT_start - 1      ; Size of GDT - 1
    dd GDT_start                    ; start of GDT


CODE_SEG equ GDT_code - GDT_start
DATA_SEG equ GDT_data - GDT_start
```

Kernel entrypoint

```
section .text
[bits 32]


_start:
[extern main]
call main                    ; calls kernel function main()
jmp $
```

Keyboard functions

```
[bits 32]

global keyboard_handler
global read_port
global write_port
global load_idt
extern keyboard_handler_main

read_port:
    mov edx, [esp + 4]
            ;al is the lower 8 bits of eax
    in al, dx   ;dx is the lower 16 bits of edx
    ret

write_port:
    mov   edx, [esp + 4]
    mov   al, [esp + 4 + 4]
    out   dx, al
    ret

load_idt:
    mov edx, [esp + 4]
    lidt [edx]
    sti                 ;turn on interrupts
    ret

keyboard_handler:
    call    keyboard_handler_main
    iretd
```

Finally we can start writing in C!

```c
/* current cursor location */
unsigned int current_loc = 0;
/* video memory begins at address 0xb8000 */
char *video_text_mem = (char*)0xb8000;


void kprint(const char *str, const char color)
{
    unsigned int i = 0;
    while (str[i] != '\0') {
        video_text_mem[current_loc++] = str[i++];
        video_text_mem[current_loc++] = color;
    }
}
```

Finally we can start writing in C!

```
extern "C" void main(){
    idt_init(); // interrupt descriptor table initialize
    kb_init(); // keyboard initialize

    textmode_print_welcome_screen();
    return;
}

/*
Interrupt from keyboard handler
*/
extern "C" void keyboard_handler_main(void){
    unsigned char status;
    char keycode;

    /* write EOI */
    write_port(0x20, 0x20);

    status = read_port(KEYBOARD_STATUS_PORT);
    /* Lowest bit of status will be set if buffer is not empty */
    if (status & 0x01) {
        keycode = read_port(KEYBOARD_DATA_PORT);
        if(keycode < 0)
            return;

        if(keycode == ENTER_KEY_CODE) {
            kprint_newline();
            return;
        }

        video_text_mem[current_loc++] = keyboard_map[(unsigned char) keycode];
        video_text_mem[current_loc++] = 0x0b;
    }
}
```

Compiling

- Kernel location is here in memory: KERNEL_LOCATION equ 0x1000
- Now we can compile and link it

```
Assembly:
    nasm "Bootloader/boot.asm" -f bin -o "Binaries/boot.bin" -i Bootloader
    nasm "Kernel/empty_end.asm" -f bin -o "Binaries/empty_end.bin"
    nasm "Kernel/keyboard.asm" -f elf -o “Binaries/keyboard.o"
    nasm "Kernel/kernel_entry.asm" -f elf -o "Binaries/kernel_entry.o" -i Kernel

Compile and link:
    i686-elf-gcc -ffreestanding -m32 -g -c "Kernel/kernel.cpp" -o "Binaries/kernel.o" -I Kernel
    i686-elf-ld -o "Binaries/kernel.bin" -Ttext 0x1000 "Binaries/kernel_entry.o" "Binaries/kernel.o" "Binaries/keyboard.o" --oformat binary

Concatenate binary files:
    cat "Binaries/boot.bin" “Binaries/kernel.bin" "Binaries/empty_end.bin" > "os_image.bin
```

Here is your first 32bit OS!