UNIFACS
Feira de Santana

Noberto Maciel
noberto.maciel@ulife.com.br

Sistemas computacionais e segurança

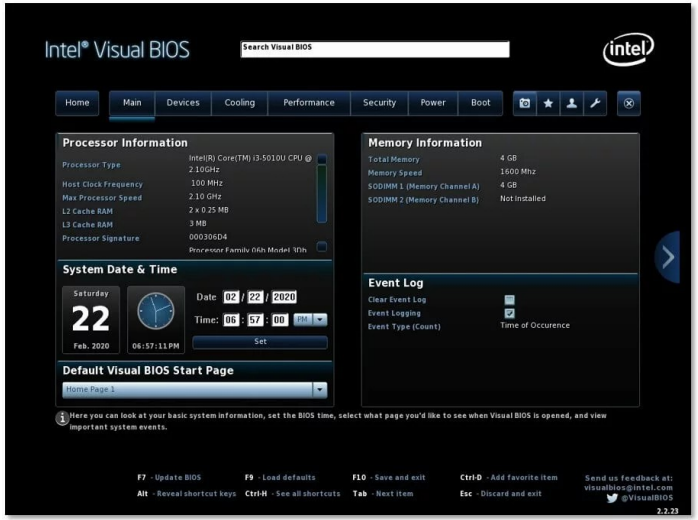
AULA 08

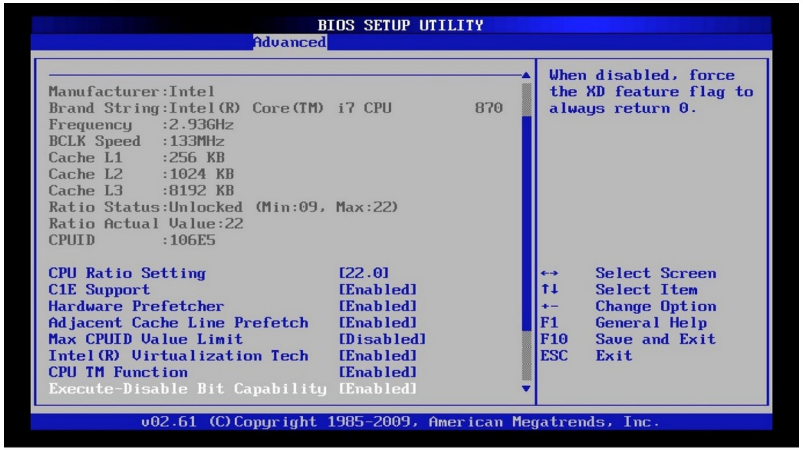
Sistemas Operacionais (continuação)

Firmware e Bootloader

Os sistemas operacionais são carregados através de um arquivo de **bootloader** que é inicializado assim que o dispositivo é ligado.

Os dispositivos inicializam o hardware por **firmware** (BIOS ou EFI) que buscam o arquivo de **bootloader** que, por sua vez, busca os arquivos de inicialização do S.O.



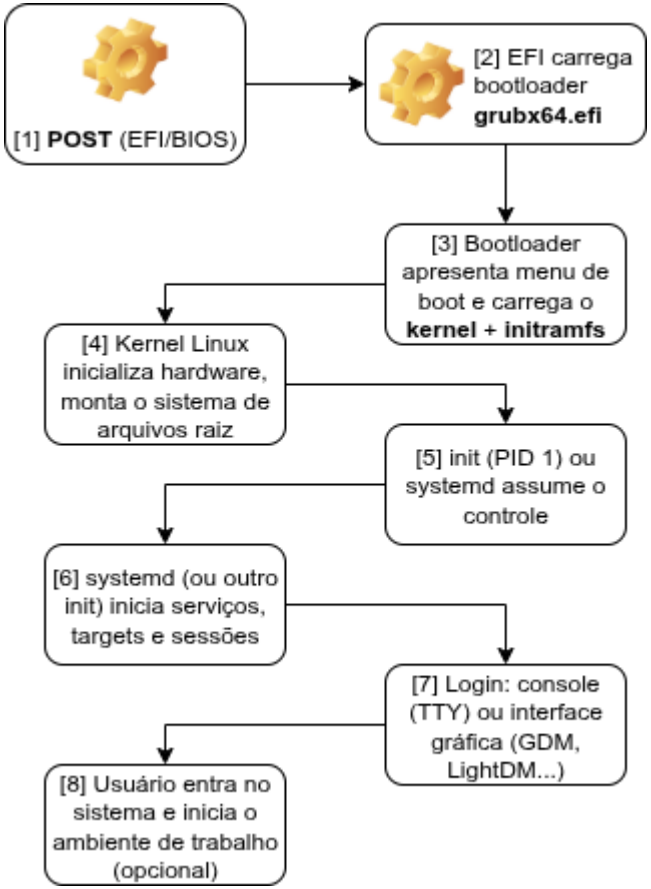


Inicialização do Linux

Após a inicialização do hardware do dispositivo, o firmware busca o arquivo de bootloader, que pode ser um arquivo **boot.bin** (binário), que é responsável por carregar o sistema na memória e inicializar o kernel do S.O.

No caso do Linux, existem vários bootloaders (GRUB, systemd-boot, LILO...). Eles possuem um arquivo de configuração (plain text – arquivo de texto) que pode ser utilizado para ajustar os parâmetros de inicialização. É nele que indicamos o(s) disco(s) de inicialização e o(s) S.O.(s).

O **bootloader** busca o arquivo de inicialização principal do Linux, o **initramfs** (e/ou o **systemd**), que inicializa todo o sistema de arquivos e demais processos.

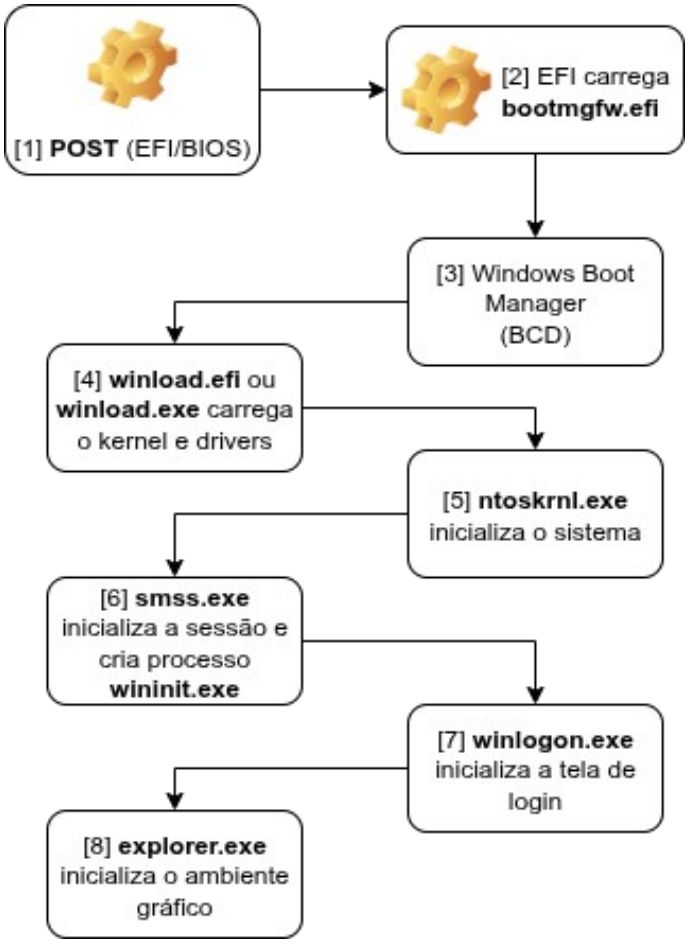


Inicialização do Windows

No Windows, nas versões atuais, o firmware, após a etapa de Self-test (POST – Power On Self-Test), inicializa o bootloader (arquivo **bootmgfw.efi**, localizado no disco em /EFI/Microsoft/Boot/), também conhecido como Windows Boot Manager, que carrega dados de incialização do Boot Configuration Data (BCD).

O Windows Boot Manager carrega na memória o arquivo de start do windows, que pode ser o **winload.efi** ou **winload.exe**.

O **winload**, por sua vez, carrega o **kernel** do Windows (**ntoskrnl.exe**), que inicializa a gerência de processos, memória, hardware, ativa os drivers do sistema, serviços essenciais e cria o processo **System**, que é o primeiro processo em espaço de usuário.



Instalando NASM e QEMU

O **NASM** é um compilador de código Assembly para binários. Sua sintaxe básica é:

nasm -f bin arquivo.asm -o arquivo.bin

O **QEMU** é um ambiente de virtualização simples. Assim como o NASM, ele não possui interface gráfica. A sintaxe básica para execução de um Sistema Operacional na VM é:

qemu-system-x86_64 [disco] [arquivoDoSO.img]

O processo é o mesmo tanto no Linux quanto no Windows, porém, no Windows, pode ser necessário configurar as variáveis de ambiente para que tanto o nasm quanto o qemu funcionem corretamente. Coloque na variável PATH o caminho dos dois programas.

Criando o bootloader

Primeiro, é necessário desenvolver o arquivo de inicialização, no nosso caso, será o **boot.asm**.

Este arquivo indica a localização do kernel do seu sistema operacional e faz o seu carregamento na memória.

As mensagens de inicialização são impressas na tela e o disco onde se encontra o kernel deve ser indicado nele.

home > noberto > Documentos > UNIFACS > disciplinas > Sistemas computacionais e segurança

```
[org 0x7C00]
BITS 16
start:
    xor ax, ax
    mov ds, ax
    mov es, ax
    mov [BOOT_DRIVE], dl       ; guarda o drive de boot
    mov si, msg
print:
    lodsb
    or al, al
    jz load_kernel
    mov ah, 0x0E
    int 0x10
    jmp print
load_kernel:
    mov ah, 0x02        ; BIOS read
    mov al, 1           ; 1 setor
    mov ch, 0
    mov cl, 2
    mov dh, 0
    mov dl, [BOOT_DRIVE]
    mov bx, 0x8000
    int 0x13
    jc disk_error
    jmp 0x0000:0x8000
disk_error:
    mov si, err
err_loop:
    lodsb
    or al, al
    jz $
    mov ah, 0x0E
    int 0x10
    jmp err_loop

msg db 'Bootloader iniciado com sucesso!', 0
err db 'Erro ao carregar o kernel!', 0
BOOT_DRIVE db 0
```

Criando o kernel

O próximo passo é criar o **kernel.asm**, que é o arquivo que possui o seu sistema operacional que será carregado na memória. Nele, podemos ter os comandos básicos, a interface e a comunicação com periféricos.

Neste exemplo, o kernel apenas será carregado pelo bootloader e imprimirá uma mensagem na tela.

home > noberto > Documentos > UNIFACS > disciplinas > Sistemas computacionais e segurança (0011142) > A

```
1   ; kernel.asm - Kernel mínimo
2   BITS 16
3   ORG 0x8000
4
5   start:
6       mov si, msg          ; SI aponta para a mensagem
7
8   print_loop:
9       lodsb                ; carrega byte de DS:SI em AL e incrementa SI
10      or al, al            ; verifica se AL == 0 (fim da string)
11      jz hang              ; se zero → fim da mensagem, pula para hang
12      mov ah, 0x0E         ; função BIOS: imprimir caractere
13      int 0x10
14      jmp print_loop
15
16  hang:
17      jmp hang             ; trava aqui (loop infinito)
18
19  msg db 'Kernel rodando com sucesso!', 0
```

Compilando os arquivos

Agora vamos usar o NASM para compilar os arquivos boot.asm e kernel.asm.

Iremos fazer o mesmo processo para os dois arquivos.

nasm -f bin boot.asm -o boot.bin

nasm -f bin kernel.asm -o kernel.bin

Ao final do processo, teremos dois arquivos .bin na pasta do seu S.O.

Concatenando os arquivos binários

Neste passo, vamos concatenar os dois arquivos criados em um único que será a nossa imagem do S.O.

No windows, podemos usar o comando **copy** e no Linux, usamos o **cat**.

Portanto:

Windows: **copy /b boot.bin + kernel.bin os.img**

ou

Linux: **cat boot.bin kernel.bin > os.img**

Inicializando nosso S.O. na VM

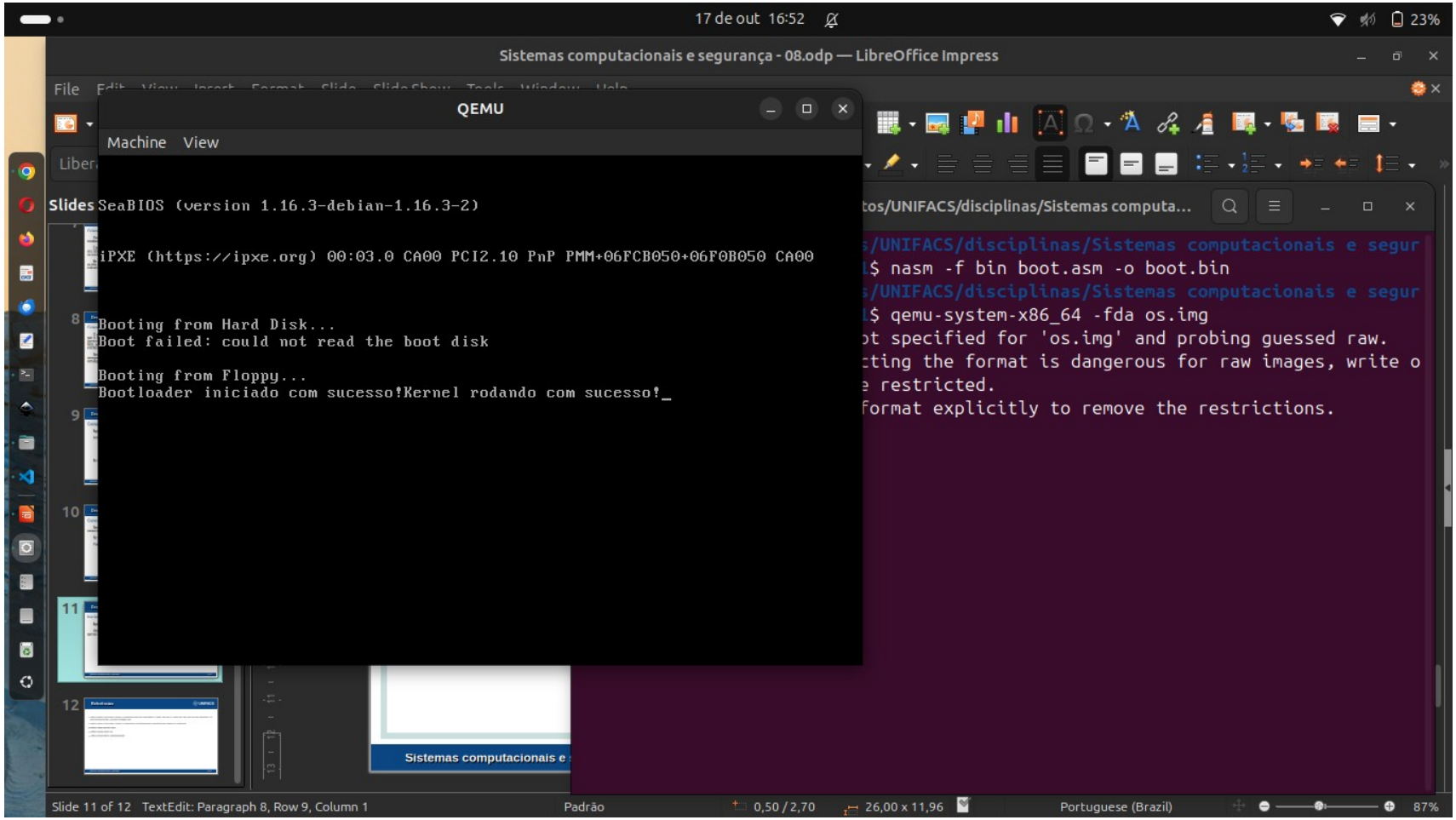
Após criarmos a nossa imagem do S.O., agora podemos inicializá-la com o QEMU.

Vamos inicializar pelo floppy disk (disco flexível, que não existe mais atualmente) apenas como forma de evitar erros, pois, esse disco é pequeno e facilmente endereçável.

O Floppy Disc A (unidade de disco A) é dada por **-fda** no comando do QEMU, portanto:

qemu-system-x86_64 -fda os.img

Desenvolvendo um S.O. simples



Considerações:

Podemos construir um prompt e comandos dentro do arquivo kernel.asm para serem executados.

A imagem do sistema operacional pode ser utilizada em qualquer VM, porém, são necessárias modificações NO CÓDIGO ASSEMBLY para adequar a área de memória e disco para inicialização.

O desenvolvimento em assembly não é foco desta UC, portanto, não trataremos sobre esta linguagem.

O código fonte dos sistemas pode ser baixado em **https://github.com/nobertomaciel/SCS-UNIFACS**.

Referências

[1]https://learn.microsoft.com/en-us/windows/security/operating-system-security/system-security/secure-the-windows-10-boot-process?utm_source=chatgpt.com

[2]https://learn.microsoft.com/en-us/windows-hardware/drivers/devtest/boot-options-in-windows

[3]https://help.ubuntu.com/

[4]https://www.nasm.us/

[5]https://www.qemu.org/download/