



Universidade Federal de Roraima
Departamento de Ciência da Computação
Sistemas Operacionais

Bootloader

BOA VISTA, RR
2021



Ramsés Messias De Oliveira Carvalho
Paulo César Pereira Belmont
Valeria Alexandra Guevara Parra

Bootloader

Trabalho apresentado como
requisito parcial para a aprovação
no componente de Sistemas
Operacionais do curso de Ciência
da Computação.
Orientador: Prof. Hebert Rocha.

BOA VISTA, RR
2021

SUMÁRIO

1. Introdução	4
1.1. Boot	4
1.2. Bios	4
1.3. Master Boot Record	4
1.4. Partição Ativa	5
1.5. Bootloader	5
2. Desenvolvimento Geral do Projeto	6
3. Testes	7
3.1. Criação do Diretório	7
3.2. Criação do Arquivo boot.asm	8
3.3. Código do Arquivo	8
3.4. Montagem do Código	9
1.5. Execução do Bootloader	9
3.6. Saída do Arquivo	9
4. Conclusão	9
5. Referências	10

1. INTRODUÇÃO

O projeto visa a construção de um bootloader utilizando o emulador de processador Qemu e o assembler Nasm, a fim de representar o funcionamento do processo de boot com kernel em Linux e tendo como sistema operacional o Ubuntu. Neste relatório ilustramos o processo de boot, o que é um bootloader e como desenvolver um.

1.1. Boot

O boot é um processo que ocorre a partir do momento em que pressionamos o botão de ligar do computador, dando assim acesso a um software presente na bios que fará uma varredura até o total carregamento do sistema operacional, este processo pode ser dividido em 4 fases principais:

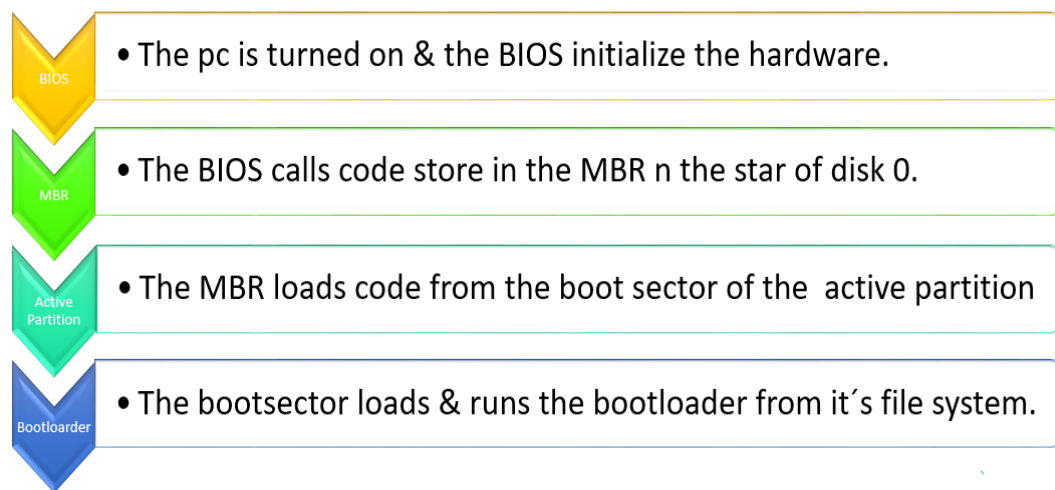


Figura 1. Etapas do boot.

1.2. Bios

Ao ligar o computador, os registrados são resetados para um valor pré-definido e as instruções contidas nestes registradores serão executadas pela CPU. assim, estas instruções apontam para a memória ROM, que contém a bios do sistema, que será executada. A partir disso, ocorrerá um processo denominado Power-On Self Teste, onde ocorrerá um teste de funcionalidade e serão inicializados os periféricos. E finalmente, um programa de boot será carregado na memória assim, a BIOS termina sua participação no processo de boot e passa o controle para o Master Boot Record.

1.3. Master Boot Record

Master Boot Record é um tipo essencial de setor de inicialização encontrado nos primeiros bytes do disco. O MBR armazena as informações sobre como as partições lógicas são organizadas nessa mídia, incluindo um sistema de arquivos, onde o bootstrap irá procurar

partições ativas e checa se as demais estão desativadas, assim finalmente passa as instruções para a CPU às executá-las.

Por fim, temos o boot signature que são 2 bytes finais do MBR, onde será apontado se o dispositivo é bootável(assume os valores 0x55 e 0xAA) ou não, se assumir valores diferentes.

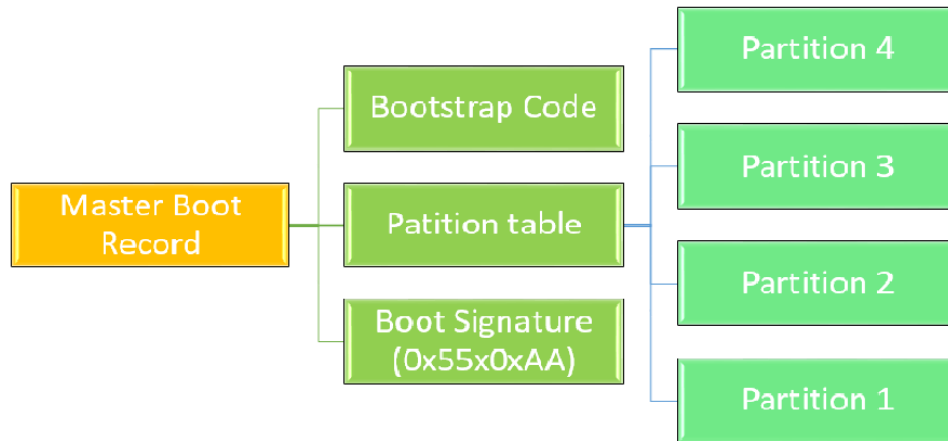


Figura 2. Particionamento do MBR.

1.4. Partição Ativa

Ao ser carregado na memória, a CPU executa o código presente no bootsector, onde irá realizar um desvio, podendo ser apenas de bytes ou de partição, e após isso , executando o bootloader, ao encontrar esse arquivo, ele executa a última instrução presente no bootsector e parte para a execução do bootloader, que irá preparar a tomada de controle do sistema operacional.

1.5. Bootloader

O bootloader será responsável por carregar toda a base de dados necessária para a tomada de controle do sistema operacional, após finalizar esses processos, o controle será passado ao SO.



Figura 3. Etapas do bootloader em Linux.

2. Desenvolvimento Geral do Projeto

O projeto segue vários passos para o cumprimento de seu objetivo, desde a instalação de um ambiente Linux até a execução do bootloader em linguagem de máquina.

Inicialmente, foi instalado um software de virtualização, permitindo a emulação de um sistema operacional com o Kernel Linux. O VirtualBox, software desenvolvido pela Oracle, foi utilizado para cumprir essa tarefa. O software foi instalado por um arquivo executável windows no site da Oracle (link nº1 nas referências).

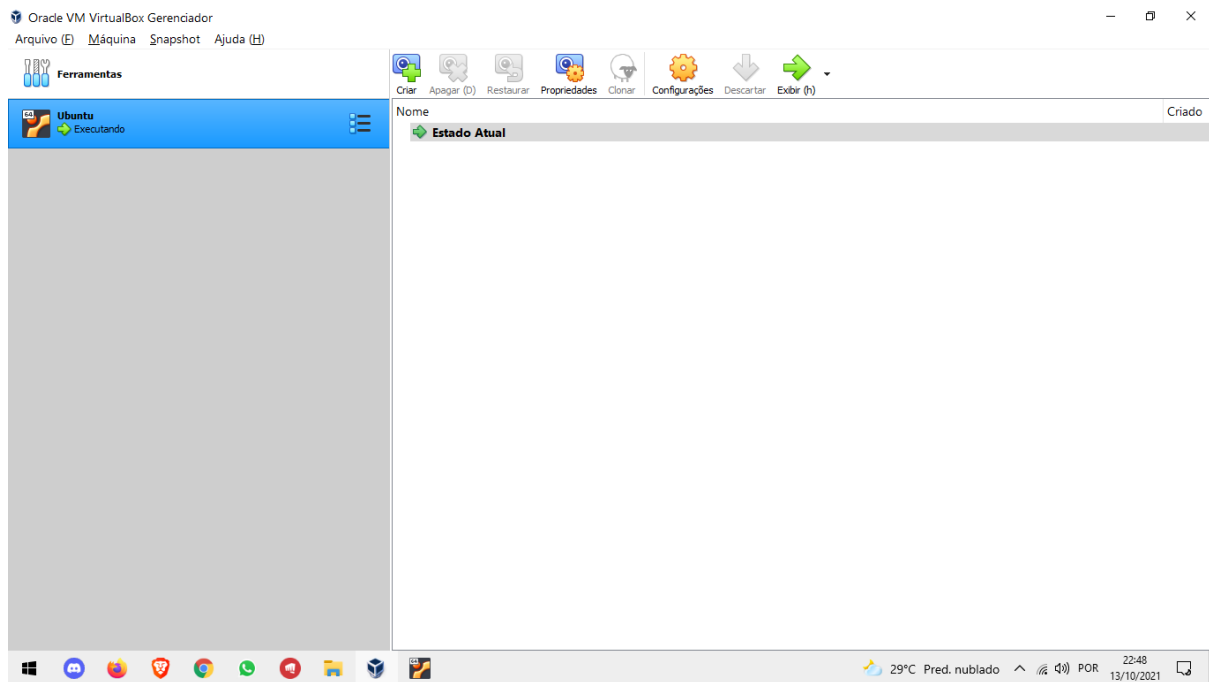


Figura 4. VirtualBox com uma Máquina Virtual Linux instalada.

Após a instalação da VirtualBox, foi montada a máquina virtual a partir do arquivo ISO da distro Ubuntu. Para esse projeto, foi utilizada a distro Ubuntu 20.04.3.

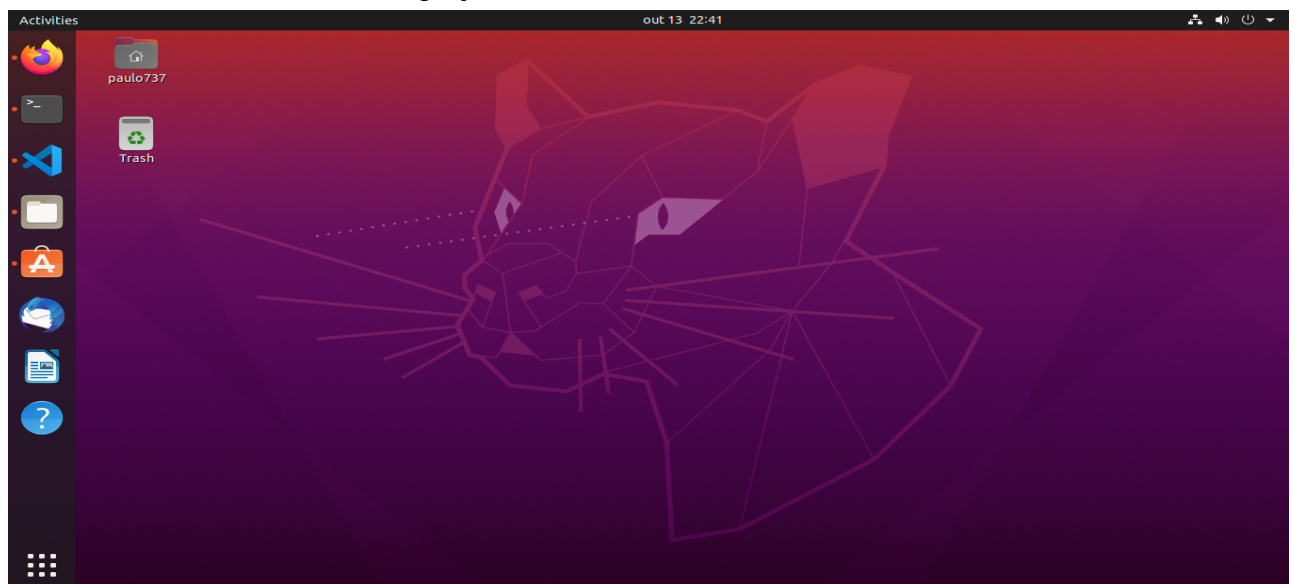
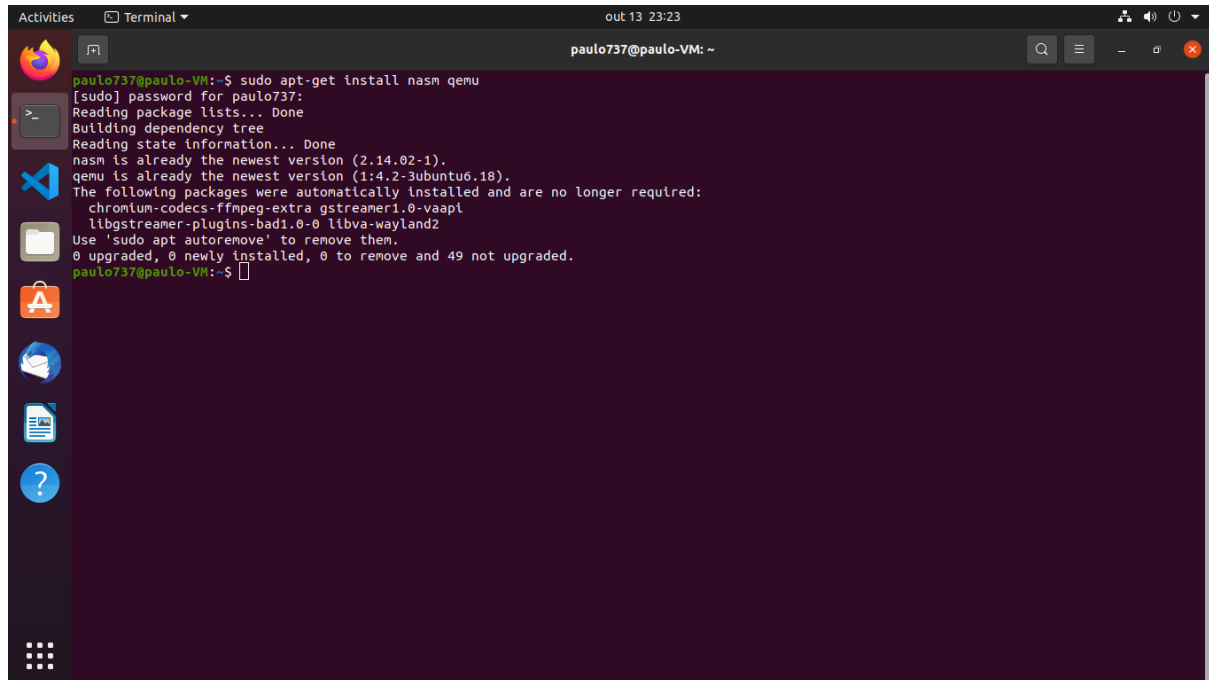


Figura 5. Ambiente Linux, distro Ubuntu emulado pelo VirtualBox

Dentro do ambiente foram instalados pelo terminal um montador (assembler) e um emulador de CPU: NASM e QEMU, respectivamente. Também foi instalado o editor de código VS Code pelo marketplace para o propósito de programar o bootloader.



```
paulo737@paulo-VM: ~$ sudo apt-get install nasm qemu
[sudo] password for paulo737:
Reading package lists... Done
Building dependency tree
Reading state information... Done
nasm is already the newest version (2.14.02-1).
qemu is already the newest version (1:4.2-3ubuntu6.18).
The following packages were automatically installed and are no longer required:
chromium-codecs-ffmpeg-extra gstreamer1.0-vaapi
libgstreamer-plugins-bad1.0-0 libva-wayland2
Use 'sudo apt autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 49 not upgraded.
paulo737@paulo-VM: ~$
```

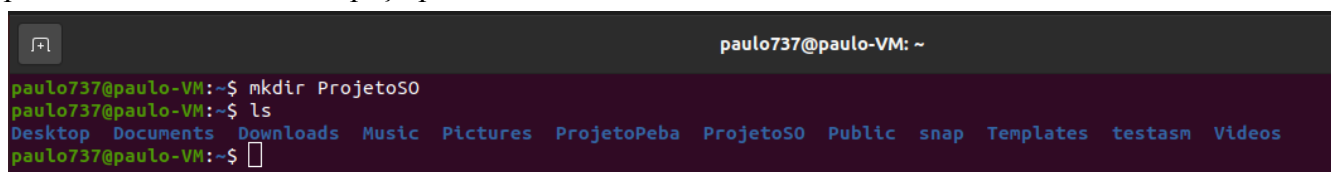
Figura 6. Comando para instalar NASM e QEMU (como já haviam sido instalados, o sistema acusou as versões dos softwares)

Sendo a programação feita no VS Code, o NASM foi responsável por montar o arquivo binário do código em assembly. Após isso, o QEMU foi utilizado para emular o arquivo binário.

3. TESTES

3.1. Criação do Diretório

O diretório do arquivo foi criado através do comando `mkdir *Nome Do Repositório*`, permitindo assim ter um espaço para armazenar o nosso bootloader.

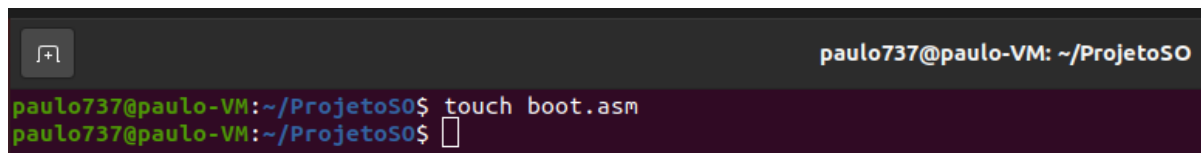


```
paulo737@paulo-VM: ~$ mkdir ProjetoSO
paulo737@paulo-VM: ~$ ls
Desktop  Documents  Downloads  Music  Pictures  ProjetoPeba  ProjetoSO  Public  snap  Templates  testasm  Videos
paulo737@paulo-VM: ~$
```

Figura 7. Criação da pasta do ProjetoSO. dá pra explicar em 10 min ez

3.2. Criação do Arquivo boot.asm(arquivo assembly)

Dentro do repositório, foi criado o arquivo boot.asm, que irá conter o código do nosso bootloader, a extensão .asm, que indica que o arquivo é de baixo nível.



```
paulo737@paulo-VM: ~/ProjetoSO
paulo737@paulo-VM:~/ProjetoSO$ touch boot.asm
paulo737@paulo-VM:~/ProjetoSO$
```

Figura 8. Criação do arquivo Boot.asm

3.3. Código do Arquivo

Em seguida, o código do bootloader em assembly foi escrito e salvo no arquivo boot.asm. O Boot é um script simples que apresenta a clássica mensagem “Hello World”. O código possui algumas partes notáveis, dentre as quais estão:

org - Especifica o endereço a ser carregado pela BIOS. 0x7C00 é o endereço padrão

bits - Define a arquitetura a ser compreendida como 16 bits

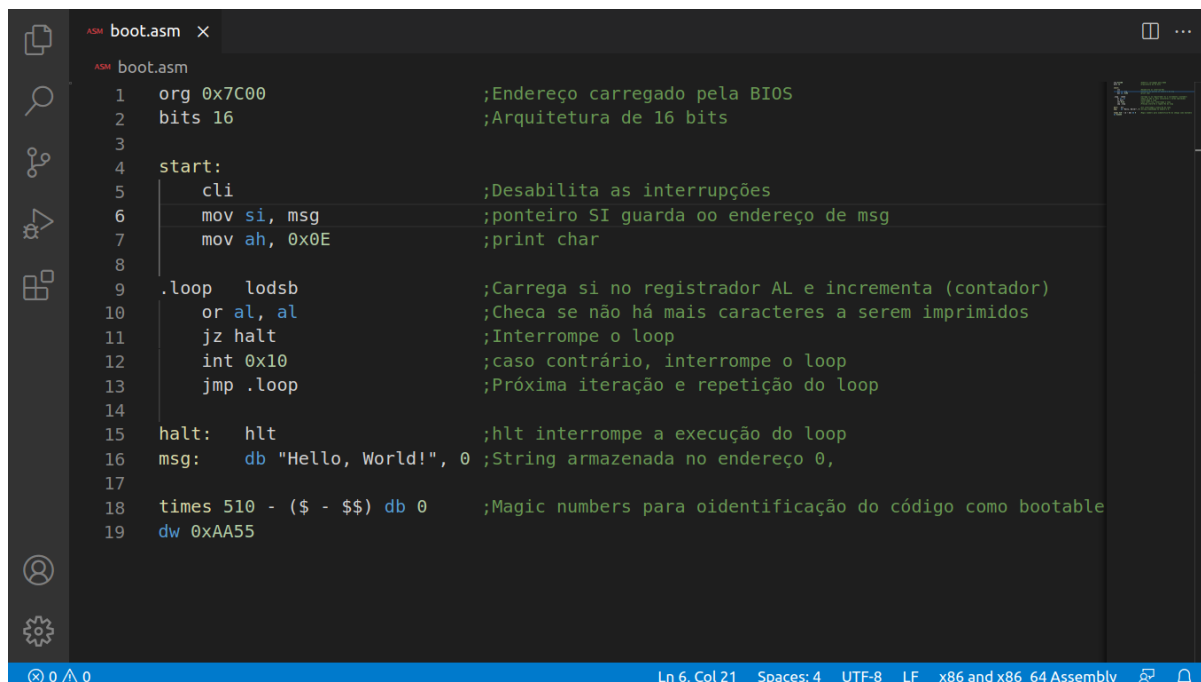
start: - Label que indica o começo do programa

.loop - Label que indica o endereço do comando a ser repetido (loop)

halt: Com o comando que cessará o loop (hlt)

msg: Armazena a string “Hello World!” no endereço 0

times 510 - (\$ - \$\$) db 0 - A parte mais importante do programa. Faz um padding para a posição do byte 510, assim os magic numbers serão escritos nos 2 últimos bytes com o comando **dw 0xAA55**.



```
boot.asm
1  org 0x7C00          ;Endereço carregado pela BIOS
2  bits 16            ;Arquitetura de 16 bits
3
4  start:
5      cli             ;Desabilita as interrupções
6      mov si, msg      ;ponteiro SI guarda o endereço de msg
7      mov ah, 0x0E     ;print char
8
9  .loop  lodsb         ;Carrega si no registrador AL e incrementa (contador)
10     or al, al         ;Checa se não há mais caracteres a serem imprimidos
11     jz halt           ;Interrompe o loop
12     int 0x10          ;caso contrário, interrompe o loop
13     jmp .loop         ;Próxima iteração e repetição do loop
14
15 halt:  hlt           ;hlt interrompe a execução do loop
16 msg:   db "Hello, World!", 0 ;String armazenada no endereço 0,
17
18 times 510 - ($ - $$) db 0 ;Magic numbers para oidentificação do código como bootable
19 dw 0xAA55
```

Figura 9. Código boot.asm.

3.4. Montagem do Código

Com o comando abaixo, o NASM recebe o arquivo “boot.asm” como argumento e o converte para um arquivo binário com o nome “boot.bin”.

```
paulo737@paulo-VM:~/ProjetoS0$ nasm boot.asm -f bin -o boot.bin
```

Figura 10. Comando para a montagem do arquivo binário.

3.5. Execução do Bootloader

O comando abaixo na imagem inicia o QEMU, que virtualiza um processador i80386 (arquitetura x86_64) e executa nesse o arquivo binário boot.bin. Após o comando, o virtualizador QEMU deve mostrar a saída “Hello World”.

```
paulo737@paulo-VM:~/ProjetoS0$ qemu-system-i386 -fda boot.bin
```

Figura 11. Comando para a execução do binário via QEMU.

3.6. Saída do Arquivo

Ao ser executado, o QEMU exibe a saída esperada, reconhecendo o código como bootável por causa de seus magic numbers.

```
SeaBIOS (version 1.13.0-1ubuntu1.1)

iPXE (http://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+07F8CB00+07ECCB00 CA00

Booting from Hard Disk...
Boot failed: could not read the boot disk

Booting from Floppy...
Hello, World!
```

Figura 12. Saída “hello, world!”.

4. Conclusão

Com este trabalho, mostramos a importância do processo de boot para um sistema computacional, assim como as etapas que compõem o funcionamento do mesmo, além disso, também demonstramos o processo de desenvolvimento de um bootloader, desmistificando as

dificuldades que envolvem a sua criação, que existem mas não chegam a ser um enigma insolúvel.

Como pode ser visto no decorrer deste relatório, existe uma intrincada gama de processos que ocorrem desde pressionar o botão de ligar do computador até a tomada de controle pelo SO, e toda esta manutenção de processos é realizada pelo bootloader.

Por fim, fica claro notar que mesmo que não seja algo extremamente complicado, o desenvolvimento de um bootloader é algo que requer uma certa quantidade de conhecimento sobre os processos que antecedem o início do SO, e outros aspectos como, controle de memória, codificação em baixo nível e entre outros. Sua importância é total para o funcionamento de qualquer aparelho computacional.

5. Referências

- Download do software VM Virtual Box
- Linux 01: My first simple bootloader;
- How to Use VirtualBox (Beginners Guide);
- How to implement your own “Hello, World!” boot loader;
- x86 Assembly - 1 - NASM, QEMU, and a Bootloader;
- Writing a Bootloader Part 1 | Alex Parker's Website.