Introdução a depuração de sistemas com softwares livres

Cesar Brod – outubro de 2014

brodtec.com | sysvale.com

Sumário

Sobre o autor

Cesar Brod atua na área de informática desde 1982. Sua formação profissional se deu em grande parte nos Estados Unidos, Canadá e França. Começou a trabalhar com o Linux em 1993 e, desde 1997 desenvolve, através da BrodTec, serviços de consultoria, gestão de projetos e criação de modelos de negócios que têm por base softwares de código aberto. Presença constante em eventos relacionados à tecnologias livres no Brasil e no exterior, Cesar foi responsável, entre agosto de 2006 e fevereiro de 2009, pela gestão de projetos dos Innovation Centers de Interoperabilidade e Open Source, apoiados pela Microsoft, na UFRGS (Universidade Federal do Rio Grande do Sul) e Unicamp (Universidade Estadual de Campinas). É o idealizador e um dos fundadores da Solis, Cooperativa de Soluções Livres e membro do projeto “Linux around the World”, de Jon “maddog” Hall, uma iniciativa internacional que busca a conscientização e disseminação do uso de software livre.

Entre os anos de 2011 e 2012 Cesar atuou como Coordenador Geral de Inovações Tecnológicas do Ministério do Planejamento do Brasil onde, entre outras funções, esteve à frente do Portal do Software Público Brasileiro e da Infraestrutura Nacional de Dados Abertos, além de ter atuado em processos de padronização de soluções.

Desde meados dos anos 1990 Cesar passou a acompanhar o Manifesto Ágil e a adotar práticas como o Extreme Programming e o Scrum para promover equipes autogerenciáveis e o desenvolvimento de projetos.

Introdução

Este livreto é parte de meu livro sobre sistemas operacionais, ainda em elaboração. O livro sobre sistemas operacionais, por sua vez é resultado de uma provocação de meu editor, o Rubens Prates. Enquanto trabalhávamos nos ajustes finais de *Aprenda a Programar – a arte de ensinar ao computador* (Novatec, 2013), ele sugeriu-me que eu usasse a mesma forma didática para trabalhar o tema de Sistemas Operacionais, tomando o Linux como base. Relutei, a princípio. Há tantos livros bons sobre o assunto que eu não acreditava na necessidade de mais um. Ao trazer o assunto para dentro de casa, minha família incentivou-me dizendo: “Simples! Faça com que o seu livro não seja só mais um.” Não foi, é claro, tão simples assim, mas você está lendo esta introdução justamente porque topei a parada.

Sou virginiano. Apesar de não acreditar em astrologia, tenho que admitir que sou a instância perfeita de todos os atributos e métodos da classe que define meu signo. Encaro um livro como um projeto ágil, com o *product backlog* definido na forma de seu índice onde, junto com os assuntos que domino, coloco outros sobre os quais ainda tenho que aprender mais antes de ousar escrever sobre eles. Desta forma, desafio-me com o prazer de sempre aprender algo novo intercalado com a exposição daquilo que já sei. Neste processo sempre descubro que ainda tenho que aprender muito mais sobre o que julgo saber para, assim, poder transmitir o conhecimento de forma simples. Dizia Einstein que, se não conseguimos explicar algo de maneira simples, é porque nós mesmos não entendemos direito esse algo.

Na construção do índice, com o nome dos capítulos que seriam posteriormente recheados, baseei-me em duas coisas: ementas da disciplina de sistemas operacionais de vários cursos de tecnologia de instituições de ensino brasileiras e na maneira, como eu mesmo, aprendi sobre sistemas operacionais.

Tive a grata oportunidade de ser exposto à uma boa variedade se sistemas operacionais na minha vida profissional. Os primeiros foram os das máquinas registradoras eletrônicas programáveis da NCR, meu primeiro emprego na área de TI. Depois conheci o OS/370 da IBM e, muitos anos depois, tive a oportunidade de traduzir para o português *O Mítico Homem Mês* (Elsevier, 2009), de Fred Brooks Jr., arquiteto-chefe do sistema predecessor OS/360. Adiante conheci sistemas operacionais de processadores de comunicação e, no início dos anos 90 comecei a trabalhar com sistemas operacionais tolerantes a falhas, para ambientes de processamento paralelo massivo. Em 1993 conheci o Linux, o sistema operacional que passou a fazer parte da minha vida profissional (e pessoal) desde então.

Entre 2006 e 2009, junto com a minha sócia, Joice Käfer, desenvolvi para a Microsoft um trabalho de gestão de desenvolvimento de softwares livres interoperáveis entre as plataformas operacionais Linux e Windows. Este trabalho envolvia a coordenação do trabalho de bolsistas da Universidade Federal do Rio Grande do Sul e da Universidade Estadual de Campinas. Na época, participei de um treinamento sobre o ambiente de processamento paralelo do servidor Windows em um evento da Microsoft em Boston, nos Estados Unidos. Vários dos trabalhos desenvolvidos neste período estão em http://ndos.codeplex.com.

Da mesma forma que em *Aprenda a Programar*, minha intenção com este livro é a de que ele seja básico o suficiente para que um iniciante sinta-se a vontade em usá-lo como sua primeira fonte de informação sobre sistemas operacionais. Mas, ao mesmo tempo, este não é um livro paternalista ou condescendente. Ao contrário, minha ideia é a de que ele

constitua-se em um constante desafio intelectual, apresentando caminhos para aprofundamentos diversos que não cabem nos limites práticos de um único livro.

O conteúdo que disponibilizo agora são as partes do livro relativas a uma introdução à linguagem assembly e à depuração de programas, nessa linguagem, usando o emulador `QEMU` e o depurador `gdb`. Faço isso como incentivo àqueles que querem participar do concurso de porte e performance de módulos do kernel Linux promovido pela Linaro (performance.linaro.org). Note que esse livreto não vai ensinar tudo a você sobre a linguagem assembly, mas ele é uma introdução suave para ela e para a arquitetura de computadores. Para seguir adiante, recomendo a leitura das demais fontes citadas em Leitura Complementar, ao final deste livreto.

Criando um ambiente para desenvolvimento e testes

Você não vai ficar desligando e ligando o seu computador nas várias experiências deste livro. Você irá trabalhar em um ambiente virtualizado – uma máquina dentro da sua máquina, um computador virtual. Assim, se algo der errado – e acredite-me, algo dará errado! - você pode simplesmente destruir seu ambiente virtual e começar tudo de novo, deixando a sua máquina real intacta.

Caso você tenha chegado a este livro depois de ter lido o *Aprenda a programar – a arte de ensinar ao computador* (Novatec, 2013) ou, se de alguma outra forma, você tem conhecimento prático sobre máquinas virtuais, pode tranquilamente fazer uma leitura rápida das próximas páginas.

Acesse a página de downloads do VirtualBox em https://www.virtualbox.org/wiki/Downloads e escolha a versão apropriada, de acordo com o sistema operacional que utiliza e a arquitetura de sua máquina. Há versões para os sistemas Windows, Mac OS X, Linux e Solaris, nas arquiteturas de 32 ou 64 bits. Tipicamente, o usuário sempre sabe qual é o sistema operacional instalado na sua máquina, mas nem todos sabem se a arquitetura é de 32 ou 64 bits. Se você sabe qual é a arquitetura de sua máquina, pode pular as instruções a seguir, baixar a versão apropriada do VirtualBox e seguir adiante.

Como saber se meu Windows está rodando em 32 ou 64 bits

Segundo informações fornecidas pela Microsoft[1], fabricante do Windows, você deve seguir esses passos, dependendo da sua versão do Windows:

Computadores executando o Windows XP

- Clique em Iniciar, clique com o botão direito do mouse em Meu Computador e clique em Propriedades.
    - Se "x64 Edition" for listado em Sistema, você está executando a versão de 64 bits do Windows XP.
    - Se não ver "x64 Edition" listado em Sistema, você está executando a versão de 32 bits do Windows XP.

A edição do Windows XP em execução é exibida em **Sistema**, próximo à parte superior da janela.

Computadores executando o Windows Vista ou o Windows 7

- Clique no botão Iniciar Imagem do botão Iniciar, clique com o botão direito em Computador e clique em Propriedades.
    - Se "Sistema Operacional de 64 bits" estiver listado ao lado de Tipo de sistema, você está executando a versão de 64 bits do Windows Vista ou do Windows 7.
    - Se "Sistema Operacional de 32 bits" estiver listado ao lado de Tipo de sistema, você está executando a versão de 32 bits do Windows Vista ou do Windows 7.

A edição do Windows Vista ou do Windows 7 em execução é exibida em **Windows**

1 Fonte: http://windows.microsoft.com/pt-br/windows7/find-out-32-or-64-bit

Edition, próximo à parte superior da janela.

Computadores executando o Windows 8

Para saber se o seu computador executa o Windows 8 em 32 ou 64 bits, use essas instruções fornecidas pelo portal Meu Windows 8[2]:

- Vá até a tela inicial do seu Windows 8 e digite *Sistema*, em seguida clique em configurações para indicar o tipo conteúdo que pretende pesquisar, neste momento você verá entre os itens listados um atalho nomeado *Sistema*, clique sobre ele.
- Na tela seguinte você conseguirá visualizar qual a versão do Windows 8 está sendo utilizada na sua máquina, se é a de 32 bits ou a de 64 bits. Essa informação está na seção *Sistema*, ao lado de *Tipo de Sistema*.

Como saber se meu Mac OS X está rodando em 32 ou 64 bits

No caso do Mac a coisa é um pouco mais complicada, uma vez que o sistema fornece o nome do processador mas não informa se sua arquitetura é de 32 ou 64 bits, o que você pode conferir em uma tabela que está na página de suporte da Apple, a fabricante do Mac: http://support.apple.com/kb/HT3696?viewlocale=pt_BR&locale=pt_BR

Segundo esta página, os passos para descobrir o processador são os seguintes:

1. Selecione Sobre Este Mac no menu Apple (o ícone da maçã) na barra de menus no canto superior esquerdo e, em seguida, clique em Mais Informações.
2. Abra a seção Hardware.
3. Localize o Nome do Processador.
4. Compare o nome do seu processador com as informações abaixo para determinar se o processador do Mac é de 32 bits ou de 64 bits.

Tabela 1.1 – Correspondência entre processadores e arquitetura

2 Fonte: http://www.meuwindows8.com/windows-8-e-32-ou-64bits/

Nome do Processador	Arquitetura
Intel Core Solo	32 bits
Intel Core Duo	32 bits
Intel Core 2 Duo	64 bits
Intel Quad-Core Xeon	64 bits
Dual-Core Intel Xeon	64 bits
Quad-Core Intel Xeon	64 bits
Core i3	64 bits
Core i5	64 bits
Core i7	64 bits

Como saber se meu Linux está rodando em 32 ou 64 bits

Se você usa o Linux, certamente sabe como abrir um terminal. Faça isso e digite o comando uname -m. Se a resposta indicar algo com o número 64, sua arquitetura é de 64 bits. Exemplo:

```
$ uname -m
x86_64
```

Para instalar e rodar o VirtualBox, você deve ter, no mínimo, um computador com 2Gb de memória, mas é recomendável ter mais do que 4Gb. Além disso, você deve ter privilégios de administrador do sistema para isso. Se você já fez a instalação de qualquer outro programa anteriormente, você não deve encontrar problemas para instalar o VirtualBox. Após baixar a versão apropriada na página de downloads mencionada no início dessa seção, execute a instalação de acordo com o processo específico para o seu sistema operacional[3]. Caso você já seja usuário de alguma das distribuições Linux, há uma chance muito grande do VirtualBox estar disponível no repositório de pacotes de sua distribuição. Assim, use o procedimento adequado para a sua própria distribuição para efetuar a instalação.

Execute o VirtualBox e crie uma nova máquina virtual clicando no botão Novo que fica logo abaixo do menu superior, à esquerda. Você será conduzido a uma série de telas onde fornecerá os dados do sistema operacional que utilizará. A título de exemplo, escolhemos as seguintes opções:

Nome: Meu Sistema Operacional

Tipo: Linux

Versão: Ubuntu

Memória: 512 MB

Criar um disco rígido virtual, tipo VDI, dinamicamente alocado, 8 GB

Feito isto, inicie sua máquina virtual clicando no botão Iniciar (a seta verde que aponta para à direita, logo abaixo do menu superior do VirtualBox). Você será solicitado a dizer qual o dispositivo que vai usar para o boot, a inicialização de sua máquina. Isto porque você não tem nenhum sistema operacional ainda instalado em seu disco rígido virtual. Opte pelo drive de CD de sua máquina hospedeira, mas certifique-se de que não tem nenhuma mídia em CD ou DCD neste dispositivo. Você receberá a mensagem:

```
FATAL: No bootable medium found! System halted.
```

Pronto! Você só tem o hardware pronto para começar a fazer experiências.

---

3 A Oracle disponibiliza um excelente manual de usuário do VirtualBox (em inglês), com o total detalhamento do processo de instalação para as diversas plataformas operacionais, nesse link: http://download.virtualbox.org/virtualbox/UserManual.pdf

O bootloader

Bom, não é totalmente verdade que você só tem o hardware à sua disposição depois que você liga a sua máquina. Há um componente que está presente em todos os computadores e que é um software não tão “soft” assim: o firmware. O firmware é um programa fixo, gravado em memória não volátil (por isso ele está presente mesmo quando o computador está desligado). Tipicamente, este firmware contém rotinas bem básicas que testam a presença e a saúde de componentes de hardware e ficam na espera de que mais instruções sejam carregadas a partir de um dispositivo como um disco rígido, um CD ou DVD ou uma pendrive USB. Nos computadores pessoais este firmware é, tipicamente, conhecido como BIOS – Basic Input/Output System (sistema básico de entrada e saída). Assim, até agora, na sua máquina virtual você tem o equivalente ao hardware e o BIOS, e é o BIOS quem reclama de não ter encontrado nada para iniciar o computador.

Vamos dar então, ao BIOS, um motivo a menos para reclamar. Você deve ter em mente que quem irá processar as suas instruções será um processador, o “cérebro” de seu computador, e ele só entende da chamada linguagem de máquina. Assim, independente da linguagem na qual você escreverá os seus programas, no fim das contas o processador estará executando uma versão dos mesmos na sua própria linguagem de máquina. A tradução entre uma linguagem de programação qualquer para a linguagem de máquina é chamada de compilação.

O exercício que faremos agora requer que você instale o NASM – The Netwide Assembler. Verifique na página do projeto (http://www.nasm.us) qual é a versão mais recente e instale o programa que corresponda ao seu sistema operacional. Usuários de Linux podem instalar o NASM a partir de seus repositórios tradicionais. Usuários das distribuições baseadas no Debian (como o Ubuntu, o Linux Mint e várias outras) podem simplesmente instalar o NASM com um único comando no terminal:

```
sudo apt-get install nasm
```

Sempre que você instalar um novo software, seu gerenciador de pacotes ou o programa apt-get poderão apresentar uma série de dependências que precisam ser atendidas e sugerirão quais são elas. Você deve aceitá-las, sempre confirmando quando solicitado.

Use seu editor de texto preferido e digite o código a seguir.

```
[BITS 16]       ; Instrui ao compilador para gerar codigo para 16 bits
[ORG 0x7C00]    ; A posicao em que o codigo sera colocado na memoria


start:
    mov ax, 0 ; Reserva um espaço de 4Kbytes apos o bootloader
    add ax, 288         ; (4096 + 512)/ 16 bytes por paragrafo
    mov ss, ax
    mov sp, 4096
    mov ax, 0 ; Configura o segmento de dados para este programa
    mov ds, ax
    mov si, texto       ; Coloca a posicao do texto em si
    call imprime        ; Chama a rotina de impressao
    jmp $               ; Repetição infinita
    texto db 'Funcionou! :-D', 0
```

```
imprime:                ; Coloca na tela o texto em si
        mov ah, 0Eh     ; int 10h - funcao de impressao
.repeat:
        lodsb           ; Pega um caracter da frase
        cmp al, 0
        je .done        ; Se o caracter for 0, termina
        int 10h         ; Caso contrario, imprima
jmp .repeat
.done:
ret
times 510-($-$$) db 0   ; Preenche o restante do setor de boot com 0s
dw 0xAA55               ; Assinatura padrao de boot
```

Salve seu programa com o nome de bootloader.asm em uma pasta onde você irá guardar estes e outros programas exemplo. Caso seu editor de texto permita que você salve seu texto em múltiplos formatos, garanta que usará o formato de texto puro (ANSI, Texto Puro, Texto ou outra opção equivalente em seu editor).

Agora, na pasta onde está o seu programa, execute o NASM para transformá-lo (compilá-lo) em um código binário, executável por seu processador.

```
nasm bootloader.asm -f bin -o bootloader.bin
```

Se você digitou tudo corretamente, agora você deve ter como resultado, em sua pasta, um programa compilado chamado bootloader.bin. Antes de podermos testá-lo, lembre-se que na mensagem de erro que você recebeu do BIOS reclamava da falta de uma mídia inicializável (bootable media). Você precisa converter este arquivo executável para um formato que sirva, para a máquina virtual, como uma mídia de disquete.

No Linux, você fará isso com o seguinte comando:

```
dd if=bootloader.bin of=disquete.img count=1 bs=512
```

O dd converte o arquivo de entrada especificado pela chave if para o arquivo de saída especificado pela chave of, usando o operando bs para dizer que o arquivo será escrito em blocos de 512 bytes, copiando um byte por vez, conforme define o operando count, a partir do arquivo de origem. Estes números tem a ver com a “geometria” do disquete virtual para o qual você está copiando seu bootloader. Caso você não tenha familiaridade com o comando dd, não se preocupe em entendê-lo integralmente agora.

No Windows você deve baixar um programa como o WinImage (http://www.winimage.com/) ou outra alternativa para criar a imagem do disquete.

Um parêntesis: você já sabe que este livro adotou o Linux como o padrão para os exemplos que serão usados em nosso aprofundamento sobre sistemas operacionais. Por que não aproveitar a oportunidade, caso você não tenha feito isso ainda, de instalar o Linux em sua máquina? Há várias opções para se fazer isso. Se você tiver no mínimo quatro gigabytes de memória e oito gigabytes disponíveis de espaço em disco, crie uma nova máquina virtual no VirtualBox com no mínimo 512 Mbytes de memória, um disco de oito gigabytes e dê o boot com uma imagem de CD que você pode baixar dos vários sites das distribuições Linux. Uma lista bastante completa destas distribuições pode ser encontrada no portal DistroWatch.com. Minha recomendação pessoal é o Linux Mint em sua versão Xfce, que pode ser baixada a partir do portal LinuxMint.com e

que fornece um bom meio termo entre usabilidade e economia de recursos de máquina.

Se sua máquina tiver pouca capacidade de memória (dois gigabytes ou menos), o melhor é você instalar o Linux ao lado do sistema operacional que você já utiliza e escolher, no momento em que a liga, qual o sistema que irá usar. A maioria das distribuições Linux modernas oferecem a você esta opção de instalação a partir do boot (inicialização) através de um CD ou DVD.

Se você tiver uma máquina antiga, ou se algum parente ou amigo puder fornecer uma pra você, há muitas distribuições Linux que funcionam muito bem e máquinas mais antigas e com menos recursos. O portal The Gaea Times fez uma lista muito informativa com as dez melhores distribuições minimalistas do Linux que podem rodar em máquinas com poucos recursos, ou mesmo diretamente através de uma pendrive:

http://tech.gaeatimes.com/index.php/archive/10-best-minimal-low-footprint-linux-distros/

Se, além de tudo, o computador mais antigo que você conseguiu tem um BIOS que não permite a inicialização direta a partir de uma pendrive, eu escrevi um artigo com alguns truques que permitem que você ressuscite um computador velhinho:

http://dicas-l.com.br/brod/brod_201210261924.php

Neste ponto, você deve ter o arquivo disquete.img em sua pasta e deve dizer ao VirtualBox que ele deve ser usado como mídia para o boot de sua máquina virtual. Desligue sua máquina virtual, fechando a janela que contém a mensagem de erro que você já viu anteriormente:

```
FATAL: No bootable medium found! System halted.
```

No VirtualBox, logo abaixo no menu principal, clique no ícone Configurações, selecione Armazenamento, abaixo da lista das controladoras, use o ícone que se parece a um quadrado com um sinal de mais e clique na opção Adicionar Controladora de Disquete. Você verá, na lista das controladoras, sua nova Controladora: Floppy. Clique no disquete com um sinal de mais ao lado desta controladora e depois em Escolher disco. Navegue entre suas pastas para chegar naquela que contém o seu recém criado arquivo disquete.img e clique em Abrir.

Ainda na janela de configurações, clique na seção Sistema, do lado esquerdo da janela e verifique se a Ordem de Boot está correta, listando o Disquete como a primeira opção. Se for preciso, faça os ajustes necessários e clique em OK.

Agora, clique na seta verde, logo abaixo do menu principal do VirtualBox. Se tudo deu certo, você deve visualizar uma janela como a mostrada na figura 1.

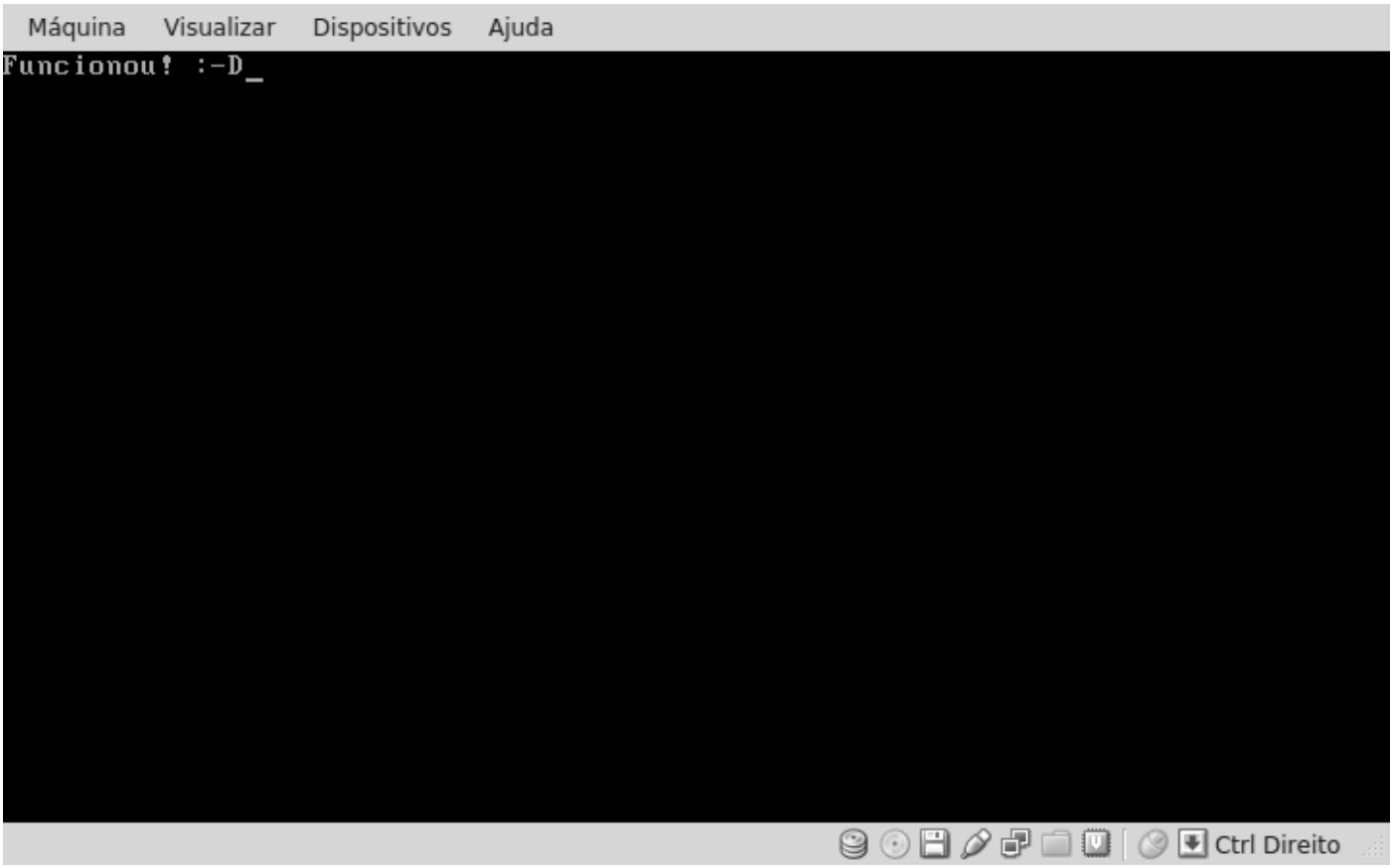

Figura 1 – Seu programa bootloader rodando na máquina virtual

O nome *bootloader* é apenas uma convenção. Traduzindo para o português, é algo como "carregador inicial" e ele é, tipicamente, o programa responsável por carregar todos os demais componentes do sistema operacional. Nada impede, como você acabou de ver, que escrevamos um programa que não faça nada. Ele será executado a partir de uma imagem de disquete desde que a última palavra do primeiro setor de 512 bytes do disquete seja 0xAA55 – o que explica a última linha de nosso código.

Se você conhece um pouco de Assembly, os comentários do programa (adaptado de um tutorial sobre o sistema MikeOS que você conhecerá melhor no próximo capítulo) que escreveu "Funcionou! :-D" na tela de seu computador devem ser esclarecedores o suficiente. Mas não se preocupe muito com isso agora. Adiante falaremos mais sobre Assembly e linguagem de máquina. Se você tiver o espírito aventureiro e gostou da brincadeira de interagir diretamente com seu computador, a melhor fonte de informações sobre o assunto é o livro *Art of Assembly Language*, de Randall Hyde (No Starch, 2010), que está disponível na íntegra no seguinte endereço web: http://www.artofasm.com

O kernel

A primeira coisa que o bootloader fará é carregar o núcleo, ou kernel do sistema operacional. É o kernel que irá fazer a interface direta com todos os componentes de hardware do sistema, com o apoio de drivers específicos para cada tipo de dispositivo e, em muitos casos, interagindo com o firmware destes dispositivos. Assim, como no caso do BIOS, vários dispositivos modernos possuem seu próprio pequeno sistema. Placas gráficas com aceleração gráfica tridimensional, por exemplo, chegam até a ter conjuntos de comandos específicos para o acesso a seus recursos, além de memória própria – sob muitos aspectos, as placas gráficas são, de fato, um outro computador dentro de seu computador.

É responsabilidade do kernel o gerenciamento do acesso à memória pelos programas aplicativos e também o seu acesso aos dispositivos de hardware. Ou seja, quando você imprime um texto, é o kernel quem intermedia o acesso de seu programa à impressora, com o uso de um driver específico para a mesma.

Ele também possui funções que garantem o melhor desempenho do processador em determinadas condições e, mesmo contando com o sistema de arquivos como o seu grande auxiliar nesta tarefa, é o kernel que organiza os dados que são armazenados em discos rígidos, pendrives ou outros dispositivos.

Um parêntesis: você já sabe que o foco deste livro é o sistema operacional Linux. Há os que preferem identificar apenas o kernel deste sistema com o nome Linux, chamando o sistema operacional completo de GNU/Linux, com o intuito de creditar as inegáveis contribuições do projeto GNU, já mencionado anteriormente, ao sistema operacional Linux. Há, inclusive, aqueles que são extremamente vocais quanto a adoção desta nomenclatura. O Linux, porém, além das contribuições do projeto GNU conta com inúmeras outras, entre elas da Berkeley Software Distribution (BSD), e das fundações Apache e X.org, mas a lista é muito mais extensa. Afinal, Linux acabou tornando-se uma “marca” para o sistema operacional livre e de código aberto construído a partir de múltiplas contribuições, todas importantes.

Além disso, na mensagem original de Linus Torvalds sobre seu projeto, em 1991, ele dizia que estava criando um sistema operacional, não apenas o kernel de um sistema operacional. Por isso, neste livro, sempre que nos referirmos ao sistema operacional completo o chamaremos de Linux e, quando nos referirmos ao núcleo do sistema operacional o chamaremos de kernel Linux ou, simplesmente, kernel.

A seguir, um trecho da mensagem de Linus traduzida para o português:

> De: torvalds@klaava.Helsinki.FI (Linus Benedict Torvalds)
>
> Newsgroups: comp.os.minix
>
> Assunto: O que você mais gostaria de ver no minix?
>
> Resumo: pequena pesquisa pra meu novo sistema operacional
>
> Data: 25 de agosto de 1991 20:57:08 GMT
>
> Organização: University of Helsinki
>
> Olá a todos usando o minix.
>
> Estou fazendo um sistema operacional livre (apenas um hobby, ele não será grande e profissional como o gnu) para clones do AT 386(486). Isto vem sendo desenvolvido desde abril e está começando a ficar pronto. Gostaria de quaisquer opiniões sobre o que as pessoas gostam e não gostam no minix, já que meu SO é um tanto similar (o mesmo leiaute físico do sistema de arquivos – por questões práticas – dentre outras

coisas).

A história do Linux, por Linus Torvalds, pode ser lida neste link:

http://www.cs.cmu.edu/~awb/linux.history.html

Tipos de kernel

Em 1992 comecei a trabalhar na Tandem Computers (hoje a divisão de sistemas NonStop da HP). Na empresa fiz cursos sobre os sistemas operacionais Unix e Guardian. Um dos livros de referência recomendado pelos instrutores era o recém-lançado *Modern Operating Systems*, de Andrew S. Tanenbaum (Prentice Hall, 1992). Hoje este livro já está em sua terceira edição e tem tradução para o português.

Por causa deste livro conheci o grupo de notícias comp.os.minix e, através dele, o Linux. O Linux permitiu-me que eu pudesse testar vários softwares disponíveis para o sistema Unix comercializado pela Tandem (baseado no Unix SVr4 da AT&T) em meu laptop antes de instalá-los nos clientes que adquiriam os computadores. Na época, a única razão de eu ter instalado o Linux, em vez do Minix, em minha máquina, foi a maior facilidade de instalação do primeiro.

A leitura das mensagens deste grupo provou ser de grande utilidade complementar aos meus estudos de sistemas operacionais na Tandem e à minha compreensão do "mundo Unix". Um marco histórico foi o debate entre o jovem Linus Torvalds e o, então já consagrado, Andrew Tanenbaum, sobre qual deveria ser o tipo ideal de arquitetura de um núcleo de sistema operacional. A Wikipedia traz uma página muito boa, em português, sobre este debate em http://pt.wikipedia.org/wiki/Debate_entre_Tanenbaum_e_Torvalds.

Andrew Tanenbaum, obviamente, defendia o microkernel, a arquitetura usada no Minix, enquanto Linus Torvalds, também obviamente, defendia a arquitetura monolítica usada no Linux.

Arquitetura monolítica

A arquitetura monolítica é mais simples, já que todas as funções do kernel estão dentro do próprio espaço do kernel. Mais adiante, quando você for personalizar o kernel do Linux isto ficará bem evidenciado. Além do Linux, sistemas operacionais como o BSD, o Solaris e as versões antigas do Windows usam este tipo de arquitetura.

Arquitetura microkernel

Na arquitetura microkernel, as funções do kernel são reduzidas ao mínimo, ficando todos os serviços do sistema no chamado espaço de usuário. O espaço do kernel é um espaço privilegiado, de contato direto com o hardware. O espaço do usuário não tem esse contato, só podendo fazê-lo através de serviços fornecidos pelo microkernel. Assim, processos do sistema de arquivos, de comunicação, tratamento de memória, todos estão no mesmo espaço de usuário. O HURD (kernel do sistema operacional GNU), o Minix e o sistema operacional do tablet Blackberry usam este tipo de arquitetura de kernel.

Arquitetura híbrida

A partir da versão Windows NT, a Microsoft passou a adotar uma arquitetura híbrida de kernel, misturando os conceitos das arquiteturas monolítica e híbrida, com alguns serviços do kernel rodando em espaço de usuário e outros em espaço do kernel. Linus Torvalds e outros defendem que esta não é uma arquitetura híbrida de fato, mas sim escolhas de levar para o espaço de usuário algumas das funções de um kernel, de fato, monolítico. Sou da mesma opinião.

A escolha entre uma arquitetura ou outra é quase pessoal, já que ambas têm suas

vantagens e desvantagens. Um microkernel é, em teoria, mais fácil de ser portado para diversos tipos de processadores, uma vez que a porção que interage diretamente com o hardware é reduzida. Um kernel monolítico tem, em teoria, maior desempenho, já que dentro dele já estão todos os componentes necessários para a comunicação com o hardware. Na prática, mesmo com um kernel monolítico, o Linux está portado para diversas plataformas.

Falando com o hardware

Antes de explorar o sistema operacional Linux, é conveniente que você conheça melhor a arquitetura de seu computador para ter uma compreensão melhor de como as coisas funcionam no limiar entre o hardware e o software. Se você já tem familiaridade com este assunto, pode tranquilamente saltar para o próximo capítulo.

A primeira coisa que você deve saber é que os componentes de um computador: seu processador, memória e dispositivos de entrada e saída de dados só entendem um tipo de informação: algo está desligado ou ligado, e isso é representado pelos números 0 e 1 respectivamente. É por isso que você ouve que um processador é de 8, 16, 32, 64 bits, que uma memória tem um barramento de 16, 32 bits e assim por diante. No fim das contas, cada um destes “bits” é um sinal que pode estar ligado ou desligado.

Um processador que tenha oito pinos para o endereçamento de memória pode acessar, diretamente, até 256 posições de memória (o zero conta como uma posição também). Cada posição de memória pode conter palavras de 8, 16, 32 bits e assim por diante. Para facilitar nossa comunicação com os componentes do computador, que só sabem contar e representar dados com os números 0 e 1, usamos tabelas de conversão e representação. A tabela 1 mostra a conversão entre números de base binária para os formatos hexadecimal e decimal.

A conversão da base binária para a decimal é feita através da soma de todos os componentes 2^Y, onde Y é a posição do bit, onde tal bit está ligado, ou seja, tem o valor 1. A posição do bit é contada da esquerda para a direita, começando por 0. Desta forma, o número 00010110 tem ligados os bits 1, 2 e 4, o que quer dizer que, em números decimais ele corresponde a 21 + 22 + 24 = 2 + 4 + 16 = 22.

A notação hexadecimal é útil já que ela agrupa quatro bits por vez, usando os números 0 a 9 e mais os caracteres A, B, C, D e F para representar 16 números (0 conta como um número) com apenas uma posição.

Você pode usar o portal http://www.binary-code.org/ para converter números binários rapidamente para o formato decimal e hexadecimal.

Tabela 1 – Conversão entre números binários, hexadecimais e decimais

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	Hex	Dec
0	0	0	0	0	0	0	0	00	0
0	0	0	0	0	0	0	1	01	1
0	0	0	0	0	0	1	0	02	2
0	0	0	0	0	0	1	1	03	3
0	0	0	0	0	1	0	0	04	4
0	0	0	0	0	1	0	1	05	5
0	0	0	0	0	1	1	0	06	6
0	0	0	0	0	1	1	1	07	7
0	0	0	0	1	0	0	0	08	8
0	0	0	0	1	0	0	1	09	9
0	0	0	0	1	0	1	0	0A	10
0	0	0	0	1	0	1	1	0B	11
0	0	0	0	1	1	0	0	0C	12
0	0	0	0	1	1	0	1	0D	13
0	0	0	0	1	1	1	0	0E	14
0	0	0	0	1	1	1	1	0F	15
1	1	1	1	1	1	1	1	FF	255

Como você já deve imaginar, cada instrução que seu processador executa também é passada a ele na forma de um número binário. Assim, se o processador receber uma instrução 00000000 ele sabe que deve fazer uma operação de soma (ADD), enquanto se receber uma instrução 00111000 ele deve comparar (CMP) dois números.

Assembly

Já houve tempos em que você efetivamente informava manualmente o código binário de operações, através de chaves liga e desliga, para que um processador as executasse. Hoje você usa montadores (*Assemblers*) que convertem os mnemônicos, pequenos conjuntos de letras que trazem algum significado e são muito mais fáceis de lembrar, nos códigos executáveis binários para o seu computador – esse conjunto de mnemônicos forma o código *Assembly*. Alguns Assemblers (por vezes chamados de Macro Assemblers) ainda permitem o uso de instruções mais completas e complexas que depois são decompostas nas operações que o processador executará. O NASM, que você conheceu rapidamente no capítulo anterior, é um Assembler.

A grosso modo, quando você escreve um programa em Assembly, você deve pensar em linguagem de máquina, mas sem a necessidade de memorizar o código binário das instruções (o Assembler fará a conversão apropriada para você). Cada arquitetura de máquina possui um conjunto de instruções e componentes específicos e, se quiser aprofundar-se nesse assunto, você deverá consultar as especificações dos processadores, normalmente encontradas na página web de seus fabricantes.

Um Assembly bastante popular é o x86, oriundo da família de processadores com o mesmo nome, derivada do processador Intel 8086, cuja fabricação teve início em 1978. A razão desta popularidade foi a adoção desta família de processadores nos computadores pessoais IBM-PC e seus clones, a partir do início dos anos 1980. O Assembly x86 estendeu-se para o x86-32 (também conhecido como IA-32), para arquiteturas de processadores de 32 bits e para o x86-64, para arquiteturas de 64 bits. As extensões mais novas são compatíveis com as anteriores – ou seja, algo que você escrever em x86 irá rodar no x86-32 e x86-64. O contrário, porém, não é verdadeiro: se você usar instruções específicas de arquiteturas com mais bits de endereçamento, elas não serão compatíveis com as de menos bits.

A seguir reproduzimos, trecho a trecho, o programa que usamos como bootloader no capítulo anterior, dissecando cada uma de suas funções e começando a ampliar no nosso cinto de utilidades com muitas ferramentas que serão úteis em nosso trabalho com sistemas operacionais e no entendimento dos mecanismos internos de um computador. Na página web que faz companhia a esse livro você terá, referente a cada capítulo, uma seção listando as ferramentas usadas.

O NASM, em sua versão atual, é capaz de gerar código de máquina a partir do Assembly x86 e do x86-32. Por uma questão de simplicidade, esta rápida introdução ao Assembly abordará apenas a versão de 16 bits.

```
[BITS 16]           ; Instrui ao compilador para gerar codigo para 16 bits
```

Esta não é uma instrução para o processador, mas para o assembler NASM. Ele lerá esta primeira linha, que inicia o programa, para saber para qual arquitetura o código será gerado. Quando encontra `BITS 16` nesta linha ele gerará o código para a arquitetura x86, de 16 bits.

```
[ORG 0x7C00]    ; A posicao em que o codigo sera colocado na memoria
```

Na arquitetura x86, o endereço hexadecimal 07C00 é onde os programas começam a ser carregados. As posições antes deste endereço estão reservadas para o BIOS.

```
start:
```

As palavras que terminam com dois pontos (:) são marcadores que referenciam rotinas ou simplesmente trechos importantes do programa. Esta linha start:, por exemplo, identifica o início do programa.

```
mov ax, 0 ; Reserva um espaço de 4Kbytes apos o bootloader
```

Este comando move, para dentro do registrador de uso geral AX, o número zero. O comentário você entenderá a seguir.

Registradores são estruturas de memória internas do processador. Os processadores da família x86 possuem os seguintes registradores:

Registradores de uso geral

Identificados pelos nomes AX, BX, CX e DX, cada um destes registradores pode armazenar um número de 16 bits, ou seja:

- números binários entre 0000000000000000 e 1111111111111111
- números hexadecimais entre 0000 e FFFF
- números decimais entre 0 e 65535

Se necessário, cada um destes registradores pode ser tratado como dois registradores de 8 bits, então identificados como AH e AL, BH e BL e assim por diante.

Registradores de índice

Os registradores SI (Source Data Index – índice de origem de dados) e DI (Destination Data Index – índice de destino de dados) apontam, respectivamente, para posições de memória onde os dados serão buscados e armazenados.

Apontador da Pilha

O registrador SP (Stack Pointer) aponta para o número de 16 bits que está no topo da pilha. E a pilha é uma memória temporária similar a uma pilha de papel no qual você só pode ver o papel que está no topo da pilha. Ela funciona como uma memória temporária e é útil, por exemplo, quando você precisa substituir os valores dos registros de uso geral por novos valores mas quer manter os valores antigos em algum lugar. É necessário lembrar, porém, que se você guarda na pilha os registros que estão em AX, BX, CX e DX na pilha, nesta ordem, ao recuperar os valores você tem que saber que eles estão na ordem reversa. Como DX foi o último colocado na pilha, ele é o que está no topo dela.

O registrador SS é o registro de segmento da pilha, ou seja, ele aponta para a área da memória que será usada para o armazenamento da pilha.

Apontador de Instrução

O registrador IP (Instruction Pointer) aponta para a posição da memória onde está a instrução que o processador está executando no momento. Tipicamente, a cada instrução executada o processador salta para a instrução na posição imediatamente seguinte na memória, mas você pode mudar o conteúdo deste registrador para que ele execute instruções a partir de outras posições, como resultado de uma condição de teste ou de uma interrupção recebida a partir de um dispositivo.

Vale também ressaltar o registrador DS, responsável por armazenar o segmento de dados que é acessado por seu programa.

Você já deve ter notado que tudo o que sucede o ponto e vírgula (;) não será interpretado pelo assembler – são apenas comentários para a leitura por humanos, com o intuito de elucidar o código.

Na linha seguinte, adicionamos o valor 288 (poderíamos ter escrito 120h para a sua representação hexadecimal) ao conteúdo do registro AX, ao qual havíamos atribuído o valor zero.

```
add ax, 288             ; (4096 + 512)/ 16 bytes por paragrafo
```

Mais adiante, quando falarmos sobre gerenciamento de memória no capítulo 5, isto ficará mais claro. Por enquanto, contente-se em saber que, por padrão, a memória é organizado em segmentos de 16 bytes de oito bits cada, chamados de parágrafos. Queremos reservar um espaço após o nosso programa, onde estará a nossa pilha (stack) de 512 bytes para os dados temporários e onde posicionaremos um segmento de dados de 4Kbytes (4096 bytes). Somamos os dois e dividimos por dezesseis para saber quantos segmentos (em parágrafos de 16 bytes) são necessários.

Vamos botar a mão na massa para entender melhor como isso funciona. A partir deste momento, presumimos que você já tem uma distribuição Linux instalada em seu computador, preferencialmente a Linux Mint 17 Xfce (ou mais nova) que foi usada como base em todos os testes e exemplos deste livro, independente de você ter optado por utilizá-la dentro de uma máquina virtual, em uma instalação com “dual boot” ou como o sistema operacional padrão de seu computador.

Se você não tem muita familiaridade com o Linux, a editora Novatec publicou o excelente Descobrindo o Linux (http://www.novatec.com.br/livros/linux3/), de autoria de João Eriberto Mota Filho. O livro tem quase mil páginas que dissecam o Linux, desde seus conceitos básicos até aspectos avançados de administração do sistema. Uma ótima fonte de informação na web, totalmente em português, é o Guia Foca GNU/Linux (http://www.guiafoca.org).

Você precisará instalar o editor e visualizador de executáveis hte. Isto pode ser feito com o uso do gerenciador de pacotes de sua instalação do Linux ou, para as distribuições baseadas em Debian (como o Linux Mint, Ubuntu e outras), com o seguinte comando:

```
sudo apt-get install ht
```

Sim, o nome do pacote que instala o hte é mesmo ht. Agora, use o hte para ler os conteúdos do arquivo bootloader.bin que você criou no capítulo anterior.

```
hte bootloader.bin
```

O resultado deve ser similar ao da figura 2.

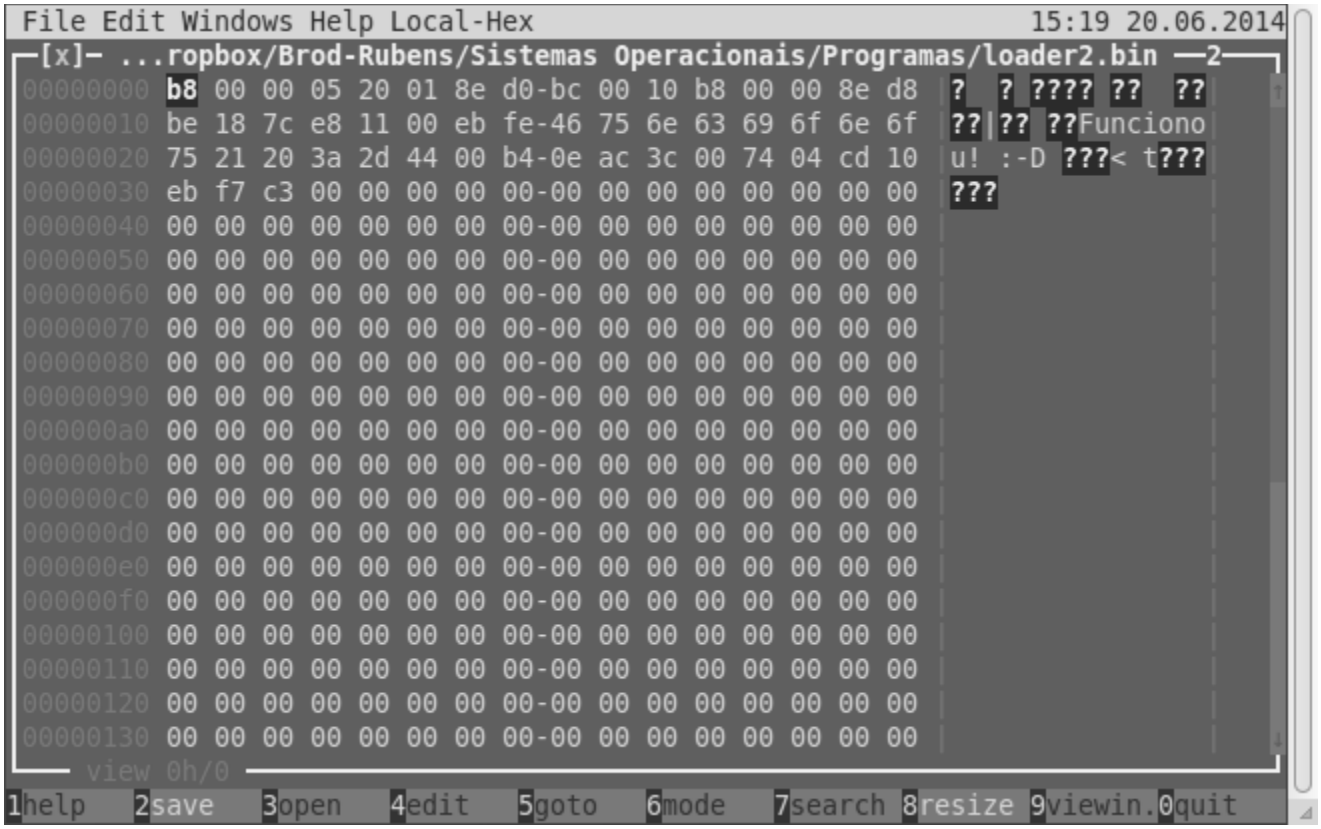

Figura 2 – Tela do editor hte exibindo os conteúdos do arquivo bootloader.bin

Note que, em cada linha, você tem representado um segmento, um parágrafo de 16 bytes. Bem à esquerda você tem o endereço inicial do segmento, seguido pelos 16 bytes de conteúdo e terminando com a interpretação que o hte tentou fazer de cada um dos bytes.

De imediato, você deve ter reparado na frase “Funcionou! :-D” mas, observando mais atentamente, no segundo e no terceiro byte você tem 00, 00. Mais adiante, na mesma linha, a sequência 00, 00 se repete e, em ambos os casos, ela aparece precedida de b8.

```
00000000 b8 00 00 05 20 01 8e d0-bc 00 10 b8 00 00 8e d8 |???? ???? ??????
```

Como o número depois desta sequência é variável, há uma boa probabilidade que B8 seja o código hexadecimal que diz ao processador para carregar o valor 0000 no registrador AX. Note que, abaixo da janela, há um conjunto de opções que podem ser acessadas a partir das teclas de função F1 a F10 (dependendo de seu emulador de terminal, F10 pode ter alguma outra função designada pela janela que o contém e não irá surtir o efeito desejado de fechar o hte – neste caso, use a combinação de teclas Control e C, ou Alt + F para acessar o menu superior e escolher a opção Quit para sair do programa). Use a tecla F6 para selecionar a visualização no formato disasm/x86 (disassembler da arquitetura x86) e F8 para usar o modo 16 bits. A figura 3 mostra que suas suspeitas estão comprovadas.

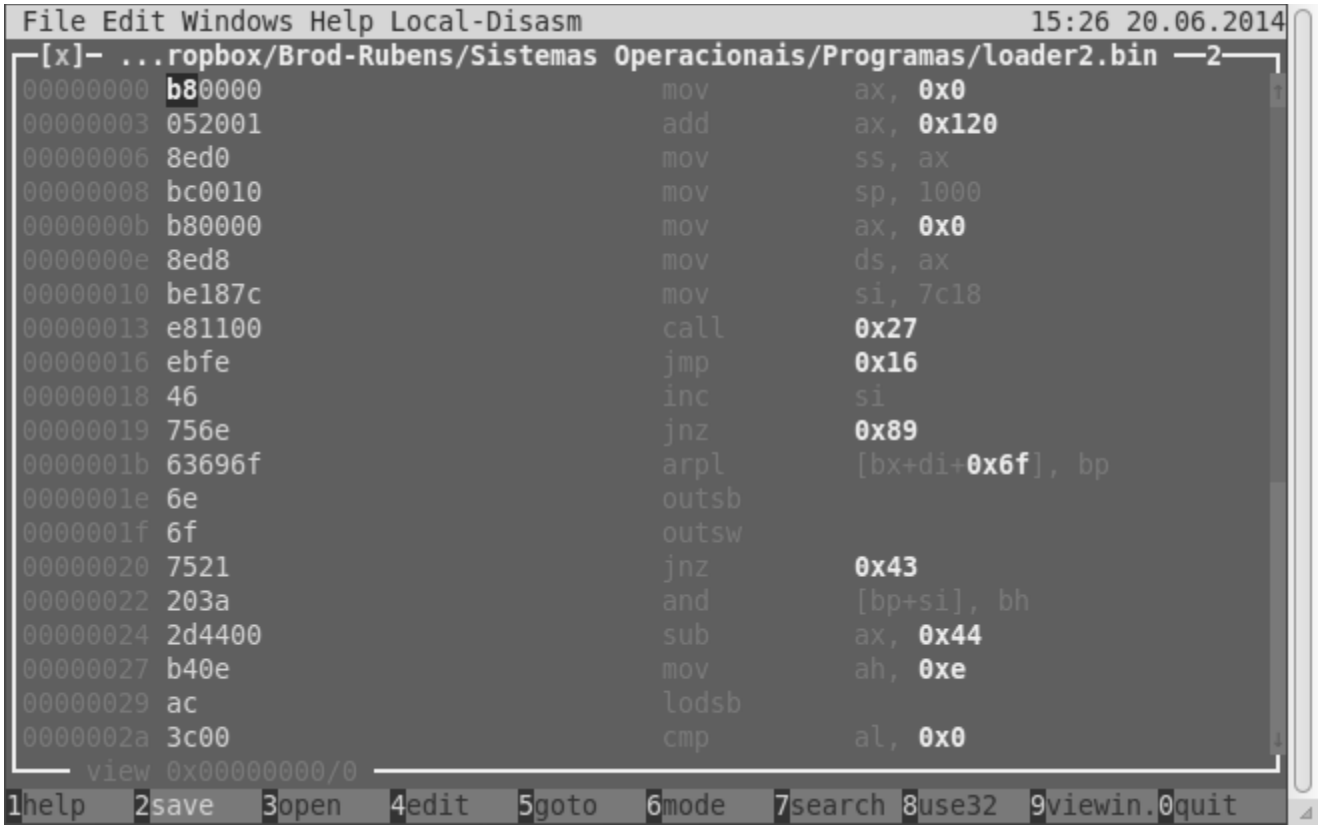

Figura 3 – A engenharia reversa do código binário do arquivo bootloader.bin no hte

Você já sabe que, no final das contas, seu processador recebe estas instruções em código binário e que, depois que o BIOS carregar este programa na memória, o endereço 07C0 (11111000000) conterá a instrução B8 (10111000). Lembre-se que você especificou esse endereço para o assembler com a instrução a seguir:

```
[ORG 0x7C00]        ; A posicao em que o codigo sera colocado na memoria
```

Um conjunto muito grande de circuitos lógicos, neste momento, será configurada para carregar no registrador AX o valor 07C0 (11111000000) e o apontador de instrução seguirá para o endereço onde encontra-se a instrução seguinte. Na execução do programa, este endereço será 07C0 somado ao endereço que o hte mostra na coluna à esquerda para esta instrução: 07C0 + 0003 = 07C3, em base hexadecimal.

Como dissemos antes, na linha a qual estamos agora, poderíamos representar 288 (o número de setores que reservamos para nossos dados temporários e segmento de dados) no formato hexadecimal. É assim que o hte mostra esta linha para nós (add ax, 0x120), usando o caractere x apenas para uma desambiguação visual explícita. O número 07c0 exibido pelo hte é, obviamente, hexadecimal, contém o caractere c. Já 0120 poderia ser interpretado como número decimal e, por isso, o x é usado apenas para explicitar que ele é hexadecimal.

O valor que está no registrador AX, quando o programa chega a este ponto é a soma dos números hexadecimais 07C0h e 0120h (288 em decimal). Se você não tem familiaridade com as notações binárias e hexadecimais a prática no mundo dos sistemas operacionais fará com que você a adquira. Por algum tempo, talvez você queira usar um conversor (como o que está neste link http://www.mathsisfun.com/binary-decimal-hexadecimal-converter.html) para fazer seus cálculos sempre com os números convertidos para o sistema decimal e, depois, vertê-los novamente para os números hexadecimais.

Dito isto, não é nada tão complicado fazer somas com números em outras bases numéricas. Lembre-se das aulas de matemática do ensino fundamental, quando você começou a fazer contas com mais de dois ou três dígitos e usava a técnica do “sobe um”, como ilustra a tabela 2, que mostra a soma dos números 137 e 204.

Tabela 2 – Soma de números de base decimal

Sobe um		1	
Número 1	1	3	7
Número 2	2	0	4
Soma	3	4	1

Lembrando a técnica, dentro de cada caixa só cabe um dígito. Assim, cada número que somado a outro resulte em um número maior que dez, “sobe um” na caixa que corresponde ao dígito da esquerda.

Com números binários é exatamente a mesma coisa, só que os “uns” sobem com mais frequência, já que 1 + 1 = 10 (sendo 10 em binário o equivalente ao número 2 em decimal). Vamos somar os números 14 e 9, respectivamente 1110 e 1001 em binário na tabela 3.

Tabela 3 – Soma de números de base binária

Sobe um	1				
Número 1		1	1	1	0
Número 2		1	0	0	1
Soma	1	0	1	1	1

Confira o resultado convertendo a soma, novamente, para um número decimal:

$2^0 + 2^1 + 2^2 + 2^4 = 1 + 2 + 4 + 16 = 23$

A soma de números em base hexadecimal também é fácil. Sobe um quando um número chega a F (15 em decimal). Aí é só contar silenciosamente de A até F quando o número decimal equivalente vai de 10 à 15. A prática torna isso, rapidamente, muito fácil. A tabela 4 mostra a soma que representa o conteúdo de nosso registrador AX até o momento.

Tabela 4 – Soma de números de base hexadecimal

Sobe um					
Número 1		0	7	C	0
Número 2		0	1	2	0
Soma		0	8	E	0

Seria bom verificar se nossos registradores estão se comportando exatamente da forma esperada. Com um pouco mais de trabalho nós podemos fazer isso. Você precisará de

duas ferramentas adicionais: o depurador de código gdb (http://www.sourceware.org/gdb/) e o emulador de processadores QEMU (http://wiki.qemu.org/). Instale-os como de costume, usando o gerenciador de pacotes de sua distribuição Linux ou, caso esteja usando a distribuição recomendada (Linux Mint 15 Xfce) ou qualquer outra baseada em Debian, com o seguinte comando:

```
sudo apt-get install qemu gdb
```

Como de costume, aceite a instalação de quaisquer prerrequisitos.

Um parêntesis: você pode, neste momento, achar que está recebendo mais informações do que tem capacidade de processar, especialmente se não teve uma exposição muito grande à aritmética com outras bases numéricas, eletrônica digital e programação. A forma como escrevi este livro é muito calcada na experiência que tive com equipes de suporte, desenvolvimento e estudantes com os quais tive a grata oportunidade de trabalhar. Isto tudo somado à própria metodologia que fui desenvolvendo, ao longo de muitos anos, para a minha própria aprendizagem continuada.

Entre 1999 e 2006 trabalhei diretamente com um grupo de excelentes profissionais e, simultaneamente, estudantes, que foram (e muitos ainda são) a base da Solis, Cooperativa de Soluções Livres (http://solis.coop.br).

Em várias ocasiões, participei com estes amigos de “sessões de estudo” em finais de tarde, na própria sede da Cooperativa. Ao final de uma dessas sessões o Nasair Júnior da Silva e a Joice Käfer (que tornou-se minha sócia em 2006) disseram-me que, simplesmente ao mostrar como um microcontrolador “pensava”, propondo um pequeno jogo onde as pessoas representavam o ambiente de execução de um programa, assumindo o papel de registradores e índices, substituindo ou incrementando seus valores, muitas coisas ficaram claras para eles.

Em muitos outros momentos percebi que apresentar perspectivas práticas, mesmo que elas possam parecer complexas a princípio, é como apresentar pequenas chaves que faltavam para uma compreensão teórica que parecia estar bloqueada e, com isso, permitir a consolidação de conhecimentos básicos que permitem novos passos adiante. Assim, respire fundo a cada nova ferramenta que este livro propõe. Dentre todas as apresentadas, você descobrirá quais serão mantidas em seu cinto de utilidades e quais você poderá deixar guardadas em sua garagem, sabendo que elas estão lá para quando você precisar. Agora fecho o parêntesis e vamos adiante!

O QEMU, assim como o VirtualBox que você já instalou e usou, é um emulador de processadores e computadores. Por ter suas raízes em ambientes de apoio a experimentação e desenvolvimento, o QEMU pode não ter uma interface muito amigável ao usuário final, mas esta interface é bastante completa e, em conjunto com o depurador gdb, permitirá a você a execução passo-a-passo de um programa, instrução por instrução, permitindo que você tire uma radiografia de todos os componentes internos importantes de sua máquina. Agora que você já tem ambos instalados, vamos seguir com nossas experiências, descobrindo se o valor no registrador AX mudou, até agora, de acordo com o esperado.

Para começar, vamos simplesmente rodar o QEMU para que ele inicie uma máquina com

a nossa imagem de disquete. Isto é feito através da linha de comando.

```
qemu-system-i386 -fda disquete.img
```

Claro que, neste caso, o QEMU deve ser executado na mesma pasta onde está a imagem de disquete que geramos. Você deve ter visto uma janela, com a sua máquina emulada, abrir com a informação exibida na figura 3 (bem similar à que você viu quando usou o VirtualBox):

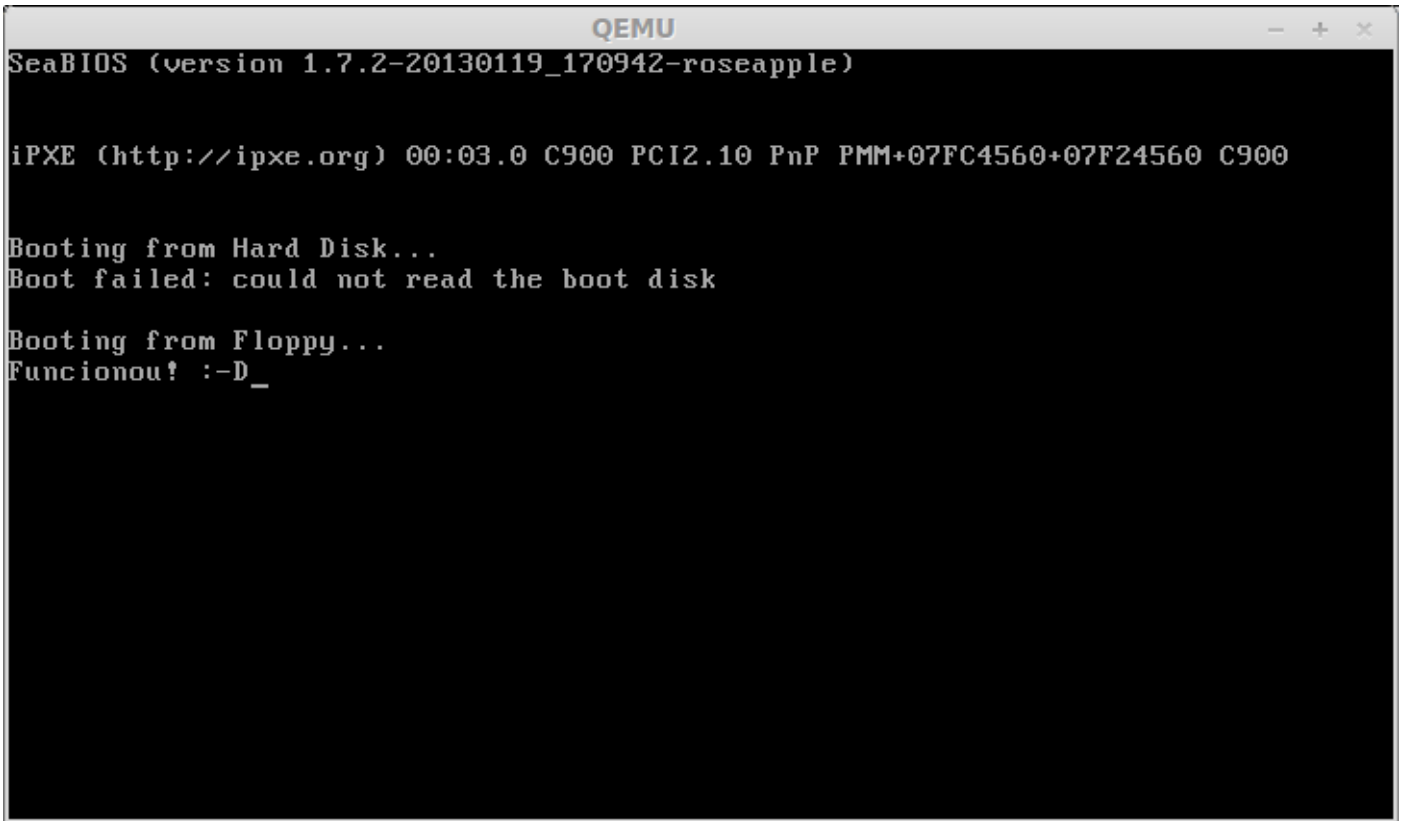

Figura 3 – O QEMU inicializado com o bootloader na imagem de disquete

Primeiro, repare na linha de comando que usamos para chamar o QEMU – qemu-system-i386 – o que significa que estaremos emulando um sistema completo, incluindo disquete, disco rígido e outros componentes. O programa responsável apenas pela emulação do processador é o qemu-i386 e ele é apropriadamente usado pelo qemu-system-i386. Para saber quais outros processadores e sistemas o QEMU é capaz de emular, na linha de comando digite qemu e, sem digitar espaço ou teclar Enter após o nome, pressione duas vezes a tecla Tab. Você deve observar um resultado similar ao seguinte:

```
qemu                qemu-ppc64abi32     qemu-system-mipsel
qemu-alpha          qemu-s390x          qemu-system-or32
qemu-arm            qemu-sh4            qemu-system-ppc
qemu-armeb          qemu-sh4eb          qemu-system-ppc64
qemu-cris           qemu-sparc          qemu-system-ppcemb
qemu-i386           qemu-sparc32plus    qemu-system-s390x
qemu-img            qemu-sparc64        qemu-system-sh4
qemu-io             qemu-system-alpha   qemu-system-sh4eb
qemu-m68k           qemu-system-arm     qemu-system-sparc
```

```
qemu-make-debian-root    qemu-system-cris         qemu-system-sparc64
qemu-microblaze          qemu-system-i386         qemu-system-unicore32
qemu-microblazeel        qemu-system-lm32         qemu-system-x86_64
qemu-mips                qemu-system-m68k         qemu-system-xtensa
qemu-mipsel              qemu-system-microblaze   qemu-system-xtensaeb
qemu-nbd                 qemu-system-microblazeel qemu-unicore32
qemu-or32                qemu-system-mips         qemu-x86_64
qemu-ppc                 qemu-system-mips64
qemu-ppc64               qemu-system-mips64el
```

O QEMU é capaz de emular uma ampla variedade de processadores e sistemas neles baseados, auxiliando muito no desenvolvimento de sistemas operacionais (ou quaisquer outros tipos de software) que possam rodar nas mais diversas configurações. Se você executar qemu-system-i386 -help, terá uma lista da ampla variedade das opções que o programa permite. Pagine esta lista encadeando-a para um comando more ou less e use um pouco de seu tempo para averiguar as opções.

```
qemu-system-i386 -help | more
qemu-system-i386 -help | less
```

Quando você encadeia a saída para o comando more, você passa para as páginas seguintes pressionando a barra de espaços. O comando less permite mais interatividade, permitindo a paginação abaixo e acima com as teclas Page Up, Page Down, assim como as setas para cima e para baixo. Além disso, você pode fazer buscas textuais dentro de uma paginação com o less teclando a barra (/) e digitando um trecho da palavra que está procurando. Use /fda e tecle Enter para descobrir o que é a chave -fda que você usou no comando com o qual evocou o QEMU. Para sair de uma paginação com o comando less, tecle a letra q.

Você usou a chave -fda para carregar a imagem de disquete (disquete.img) no primeiro disquete do sistema, mas certamente observou, na janela de emulação do QEMU, que a primeira opção do emulador era o disco rígido, o que primeiro gerou um erro, fazendo com que o QEMU, então, tentasse o disquete, com sucesso. Experimente usar o comando grep para filtrar a ajuda do QEMU e descobrir como alterar a ordem do boot.

```
qemu-system-i386 -help | grep boot
```

Uma solução que fará com que o sistema seja iniciado sem erros é a seguinte:

```
qemu-system-i386 -fda disquete.img -boot a
```

Agora queremos que o QEMU execute nosso pequeno programa de inicialização, passo-a-passo, para que possamos acompanhar o que acontece nos registradores. Para isso, vamos usar algumas chaves adicionais. Se você ainda está com a janela de emulação do QEMU aberta, feche-a. Na página oficial do QEMU e com o uso da chave -help você terá muito mais informações sobre o uso do programa. Desta forma, os próximos passos são um resumo do que você precisa fazer para que possamos atingir o nosso objetivo. Vamos usar o monitor do QEMU para exibirmos os registradores do processador e o gdb para configurarmos o ponto a partir do qual queremos executar nosso programa passo-a-passo. Primeiro, o comando para o QEMU, com suas respectivas chaves:

```
qemu-system-i386 -fda disquete.img -boot a -s -S -monitor stdio
```

A chave -s é uma abreviatura conveniente para -gdb tcp::1234, que instrui ao QEMU que passe informações e receba comandos de depuração do gdb através da porta TCP 1234 (você lerá mais sobre protocolos de rede e portas no capítulo 4, Rede). A chave -S faz com que a máquina inicie parada, aguardando por um comando seu para que comece a execução do

sistema. A chave -monitor sdtio abre o monitor do QEMU, através do qual você pode verificar o estado de vários componentes internos do processador, dentro do terminal através do qual você iniciou o QEMU.

Uma vez iniciado o QEMU (ele mostra a informação Paused – pausado – no topo da sua janela), abra um outro terminal onde vamos conectar o QEMU ao gdb. Neste novo terminal aberto, digite gdb e tecle Enter para iniciar o depurador e, em seguida, estabeleça a conexão com o QEMU com o seguinte comando - digite o que está à direita de (gdb):

```
(gdb) target remote localhost:1234
```

O gdb deve ser informado que a máquina que está sendo depurada tem uma arquitetura de 16 bits. Isto é feito com o comando a seguir - repare que as linhas que iniciam com (gdb) são as que contém o seu comando, as demais são a resposta do depurador.

```
(gdb) set architecture i8086
Assume-se que a arquitetura alvo é i8086
```

Você lembra que seu programa começa na posição 07C0h, antes desta posição de memória o computador executa o conjunto de instruções que compõem o BIOS. Assim, vamos avisar ao depurador que queremos que o sistema interrompa a sua execução exatamente quando o nosso programa começa a ser executado. Isto é feito com o seguinte comando, dentro da janela onde o gdb está em execução (a segunda linha é a resposta do gdb):

```
(gdb) br *0x7c00
Ponto de parada 1 at 0x7c00
```

O comando c, seguido de Enter, dentro da janela do gdb, faz com que o QEMU entenda que deve continuar sua execução, parando imediatamente antes da carga do programa que você escreveu. Antes de fazer isso, porém, no terminal onde você tem rodando o monitor do QEMU, digite o comando info registers e tecle Enter. O resultado deve ser similar ao que é exibido a seguir.

(qemu) info registers

```
EAX=00000000 EBX=00000000 ECX=00000000 EDX=00000633
ESI=00000000 EDI=00000000 EBP=00000000 ESP=00000000
EIP=0000fff0 EFL=00000002 [-------] CPL=0 II=0 A20=1 SMM=0 HLT=0
ES =0000 00000000 0000ffff 00009300
CS =f000 ffff0000 0000ffff 00009b00
SS =0000 00000000 0000ffff 00009300
DS =0000 00000000 0000ffff 00009300
FS =0000 00000000 0000ffff 00009300
GS =0000 00000000 0000ffff 00009300
LDT=0000 00000000 0000ffff 00008200
TR =0000 00000000 0000ffff 00008b00
GDT=     00000000 0000ffff
IDT=     00000000 0000ffff
CR0=60000010 CR2=00000000 CR3=00000000 CR4=00000000
DR0=00000000 DR1=00000000 DR2=00000000 DR3=00000000
DR6=ffff0ff0 DR7=00000400
EFER=0000000000000000
FCW=037f FSW=0000 [ST=0] FTW=00 MXCSR=00001f80
FPR0=0000000000000000 0000 FPR1=0000000000000000 0000
```

```
FPR2=0000000000000000 0000 FPR3=0000000000000000 0000
FPR4=0000000000000000 0000 FPR5=0000000000000000 0000
FPR6=0000000000000000 0000 FPR7=0000000000000000 0000
XMM00=00000000000000000000000000000000 XMM01=00000000000000000000000000000000
XMM02=00000000000000000000000000000000 XMM03=00000000000000000000000000000000
XMM04=00000000000000000000000000000000 XMM05=00000000000000000000000000000000
XMM06=00000000000000000000000000000000 XMM07=00000000000000000000000000000000
(qemu)
```

Você consegue localizar, nesse resultado, os registradores que usou em seu programa? Para ajudar, use uma tabela para anotar os valores que estão nos registradores, na medida em que executa, passo-a-passo, o seu programa. Anote os valores para a posição inicial (Paused) e após você teclar `c` (de continue) no prompt do gdb.

Bootloader	QEMU	Início	7C00h
ax	EAX	00000000	0000aa55
ss	SS	0000 00000000 0000ffff 00009300	0000 00000000 0000ffff 00009300
sp	ESP	00000000	00006f20
ds	DS	0000 00000000 0000ffff 00009300	0000 00000000 0000ffff 00009300
si	ESI	00000000	00000000
ah	EAX(H)	0000	0000
ip	EIP	0000fff0	00007c00

A posição dos registradores no ponto de parada 7C00h, que você definiu no QEMU, é a posição em que o BIOS nos deixa antes do nosso programa ser iniciado – repare no registrador `ip`, o apontador de instrução). O que vamos fazer, agora, é anotar o que acontece com os registradores relevantes, na medida em que o nosso bootloader é executado. No terminal onde você está rodando o gdb, execute o comando `stepi` (*step instruction* – dê um passo para a próxima instrução). Você deve observar algo parecido com este:

```
(gdb) stepi
0x00007c03 in ?? ()
```

Seu programa executou uma instrução e avançou para a posição de memória seguinte (7C03h). Vamos ver o que aconteceu com os registradores? Você já sabe como fazer isso: no monitor do QEMU use o comando `info registers`.

Bootloader	QEMU	7C00h	Passo 1
ax	EAX	0000aa55	00000000
ss	SS	0000 00000000 0000ffff 00009300	0000 00000000 0000ffff 00009300
sp	ESP	00006f20	00006f20
ds	DS	0000 00000000 0000ffff 00009300	0000 00000000 0000ffff 00009300
si	ESI	00000000	00000000
ah	EAX(H)	0000	0000
ip	EIP	00007c00	00007c03

Repare que os únicos registradores que mudaram foram o ip, que agora aponta para o endereço da próxima instrução a ser executada (7C03) e o ax, pois ele acaba de receber o valor 0, como exigimos que isso acontecesse na primeira linha “real” de nosso bootloader:

```
mov ax, 0
```

No gdb, use mais uma vez o comando stepi e verifique os registradores no QEMU. Repita esse processo para os próximos passos.

Bootloader	QEMU	Passo 1	Passo 2
ax	EAX	00000000	00000120
ss	SS	0000 00000000 0000ffff 00009300	0000 00000000 0000ffff 00009300
sp	ESP	00006f20	00006f20
ds	DS	0000 00000000 0000ffff 00009300	0000 00000000 0000ffff 00009300
si	ESI	00000000	00000000
ah	EAX(H)	0000	0000
ip	EIP	00007c03	00007c06

Como já havíamos discutido antes, agora adicionamos o valor `288` (poderíamos ter escrito `120h` para a sua representação hexadecimal) ao conteúdo do registrador `AX`, ao qual havíamos atribuído o valor zero.

```
add ax, 288
```

Repare que o apontador de instrução já está com o endereço da próxima.

Bootloader	QEMU	Passo 2	Passo 3
ax	EAX	00000120	00000120
ss	SS	0000 00000000 0000ffff 00009300	0120 00001200 0000ffff 00009300

sp	ESP	00006f20	00006f20
ds	DS	0000 00000000 0000ffff 00009300	0000 00000000 0000ffff 00009300
si	ESI	00000000	00000000
ah	EAX(H)	0000	0000
ip	EIP	00007c06	00007c08

Aqui, acabamos de mover o conteúdo do registrador geral AX para o o registrador SS (Stack Segment – segmento da pilha - `mov ss, ax`) que aponta para a área de memória reservada para ela. No próximo passo, o seguinte comando é executado:

```
mov sp, 4096
```

Veja como ficam nossos registradores.

Bootloader	QEMU	Passo 3	Passo 4
ax	EAX	00000120	00000120
ss	SS	0120 00001200 0000ffff 00009300	0120 00001200 0000ffff 00009300
sp	ESP	00006f20	00001000
ds	DS	0000 00000000 0000ffff 00009300	0000 00000000 0000ffff 00009300
si	ESI	00000000	00000000
ah	EAX(H)	0000	0000
ip	EIP	00007c08	00007c0b

Você já deve ter notado que os valores dos registradores estão no formato hexadecimal, por isso o registrador `sp` (`ESP`) guarda o valor 00001000, ou seja, o equivalente hexa ao valor decimal 4096.

O bootloader que estamos depurando é muito pequeno mas, se você decidir que, além de aprender um pouco mais sobre sistemas operacionais com a leitura desse livro, você irá, de fato, escrever ou melhorar sistemas operacionais, você irá se deparar com programas enormes e muito mais complexos. Quando isso acontecer, é bom você ter uma maior familiaridade com o QEMU e o gdb.

Por exemplo, na medida em que você depura o seu código, pode ser que você fique confuso sobre qual a instrução que o QEMU está executando em um determinado passo. No passo em que estamos agora, por exemplo, se executarmos, no QEMU, o comando x /i $eip, confirmamos a instrução que será executada, junto com o endereço de memória onde ela se encontra:

```
(qemu) x /i $eip
0x00007c0b:  mov    $0x0,%ax
```

Quando você adquire prática com o que está acontecendo nos registradores, você não precisa listar o conteúdo de todos eles. Acabamos de alterar, na instrução anterior, o

conteúdo do registrador SP (ESP para o QEMU), assim, podemos confirmar seu novo conteúdo com o mesmo comando x:

```
(qemu) x $esp
00001000:     0
```

E uma maneira conveniente de exibir o conteúdo do registrador no formato decimal é através do uso do comando print:

```
(qemu) print $esp
4096
```

Este livro não tem a mínima intenção de cobrir todos os aspectos do QEMU (e nem do gdb). Vale a pena, porém, você dar uma olhada no manual online do QEMU antes de seguir adiante: http://en.wikibooks.org/wiki/QEMU/Monitor

Agora que você já aprendeu mais algumas coisas, siga para o próximo passo de seu bootloader. No gdb, digite o comando si (uma abreviatura de stepi):

```
(gdb) si
0x00007c0e in ?? ()
```

No QEMU, confira o comando que acaba de ser executado e verifique o conteúdo do registrador que acaba de ser alterado:

```
(qemu) x /i $eip
0x00007c0e:  mov    %ax,%ds
(qemu) print $ds
0
```

Confere, certo? Você havia colocado zero no registrador ax na instrução anterior (mov ax, 0) e agora moveu o conteúdo do registrador ax para dentro do registrador ds (mov ds, ax).

Que tal você largar esse livro agora e tentar acompanhar o resto da execução do programa por sua conta? Não se preocupe se algo der errado! Você sempre pode começar tudo de novo e, claro, a partir do parágrafo seguinte, continuaremos depurando, em conjunto, nosso programa. Se você topou a proposta de experimentar sozinho, os próximos parágrafos servirão como uma revisão. Não deixe de usar a tabela para anotar a evolução dos conteúdos de seus registradores.

Uma proposta de trabalho em grupo

Em sua sala de aula ou grupo de estudos, faça com que algumas pessoas façam os papéis de registradores (incluindo a pilha e o apontador da pilha). Outra pessoa será a responsável por executar, passo a passo, o programa retirando, uma a uma, as instruções de uma caixa. Recorte quadrados de papel com todos os dados que você utilizará. Assim, na primeira instrução, o executor pegará um papel onde está escrito o número 0 (zero) e o entregará para a pessoa que representa o registrador ax.

Se você está trabalhando sozinho, use caixas (ou quadrados em um quadro branco) para representar seus registradores e papéis ou etiquetas adesivas para ir substituindo o conteúdo neles.

Esse tipo de exercício, bastante lúdico, aumenta consideravelmente a compreensão da estrutura do computador e do funcionamento de seus programas. Ao menos isso foi o que comprovou minha experiência prática.

Você voltou! Já? Que bom, vamos acompanhar o restante da execução do programa. Siga anotando os valores dos registradores principais com a instrução info registers do QEMU, especialmente se você está adquirindo, agora, experiência com a linguagem de máquina (o Assembly) e com a arquitetura de computadores. Daqui em diante vamos destacar, apenas, os registradores, instruções e posições de memória relevantes à continuidade da depuração de nosso bootloader. Vá para a próxima instrução (si) no gdb e confira o que aconteceu no QEMU:

```
(qemu) x /i $eip
0x00007c10:  mov    $0x7c18,%si
```

O registrador si, como você deve lembrar, aponta o índice da origem dos dados. Use o QEMU para descobrir o que está armazenado a partir do endereço 7C18h:

```
(qemu) x /c 0x7c18
00007c18: 'F' 'u' 'n' 'c'
(qemu) x /c 0x7c1c
00007c1c: 'i' 'o' 'n' 'o'
(qemu) x /c 0x7c20
00007c20: 'u' '!' ' ' ':'
(qemu) x /c 0x7c24
00007c24: '-' 'D' '\x00' '\xb4'
```

Usamos a chave /c para solicitar ao QEMU que exiba os conteúdos da memória em formato de caracteres. Ou seja, a partir da posição de memória 7C18h temos o texto “'Funcionou! :-D', 0” . Siga para o próximo passo e confira, no QEMU, o que está acontecendo.

```
(qemu) x /i $eip
0x00007c13:  call   0x7c27
```

O endereço 7C27h é onde o compilador colocou a nossa rotina de impressão de caracteres na tela. Dê mais um passo no gdb.

```
(qemu) x /i $eip
0x00007c27:  mov    $0xe,%ah
```

Repare que, da posição de memória 7C13h, o apontador de instrução saltou para a posição 7C27h, conforme instruído pela instrução `call` anterior. Nessa posição temos a instrução `mov ah, 0Eh`, que o compilador traduziu para `mov  $0xe,%ah`. Essa instrução chama uma subfunção dos serviços de vídeo do BIOS. Ela trabalha em conjunto com a instrução `int 10h`, que está mais adiante em nosso bootloader e é quem, efetivamente, colocará os caracteres na tela.

Já falamos sobre o BIOS, mas é bom ressaltar que ele é um sistema operacional básico que existe em qualquer computador. Ele está carregado antes do endereço 7C00h e já oferece uma série de funções que você pode utilizar em seu próprio sistema operacional, como a que estamos usando para imprimir caracteres na tela. Conheça todas as funções do BIOS 8086 nesse link:

http://www.computing.dcu.ie/~ray/teaching/CA296/notes/8086_bios_and_dos_interrupts.html

Que tal fazer, por exemplo, com que o emoticom `:-D` logo após a palavra `Funcionou` fique piscando em sua tela? Leia a lista de interrupções apresentada no link e faça suas experiências.

Os computadores modernos têm o BIOS gravado em memórias que podem ser reescritas, permitindo a sua atualização. Isso quer dizer que, além de escrever seu próprio sistema operacional, você pode escrever seu próprio BIOS. O projeto Coreboot (http://www.coreboot.org), por exemplo, visa substituir BIOS proprietários, fornecidos pelos fabricantes de processadores, por um BIOS totalmente aberto, que pode ser modificado por aqueles que o desejarem.

Como de costume, siga para a próxima instrução no gdb e acompanhe o que acontece no QEMU.

```
(qemu) x /i $eip
0x00007c29:  lods   %ds:(%si),%al
```

Note a instrução de nosso programa:

```
lodsb                   ; Pega um caracter da frase
```

Ela foi expandida para uma instrução mais completa:

```
0x00007c29:  lods   %ds:(%si),%al
```

A instrução lods é usada para processar uma cadeia de caracteres, um caractere por vez, a partir de uma fonte (a posição de memória apontada pelo registrador si) e um destino (o registrador ds) e a coloca na posição inferior do registrador ax (no caso, al).

Lembre-se, o registrador IP (EIP) armazena a instrução que será executada. Assim, faça com que o gdb avance para o próximo passo e acompanhe o que acontece com os registradores IP, DS, SI e AX. Usei o comando info registers no QEMU e, a seguir, reproduzo um extrato de sua saída, a cada passo dado (si) no gdb.

```
(qemu) x/i $eip
0x00007c29:  lods   %ds:(%si),%al
(qemu) info registers
EAX=00000e00
ESI=00007c18
DS =0000 00000000 0000ffff 00009300

(qemu) x/i $eip
0x00007c2a:  cmp    $0x0,%al
(qemu) info registers
EAX=00000e46
ESI=00007c19
DS =0000 00000000 0000ffff 00009300
```

Repare que começamos a percorrer a cadeia de caracteres, um a um, armazenados nas posições de memória indicados pelo apontador de pilha SI, os carregamos na parte inferior do registrador AX (AL – ou seja, preste atenção apenas aos números à direita de EAX) e comparamos com o número 0 (zero).

Caracteres são representados pela notação ASCII (*American Standard Code for Information Interchange* – código americano padrão para a troca de informações): http://www.asciitable.com

Assim, repare o que está armazenado a partir da posição de memória 7C18H, o código ASCII correspondente e o que é armazenado em AL, na medida em que o programa avança, preenchendo a tabela a seguir.

Memória (hex)	Caractere	ASCII	AL
7C18	F	46	0e46
7C19			
7C1A			
7C1B			
7C1C			
7C1D			
7C1E			
7C1F			
7C20			
7C21			
7C22			
7C23			
7C24			
7C25			
7C26			

No passo seguinte, como o que estava no registrador AL era 0e46, e não zero, o programa salta o endereço 7C32h e mais um passo, vemos que ele entrou na interrupção de impressão.

```
(qemu) x/i $eip
0x00007c2e:  int    $0x10
```

Agora, durante vários passos, as instruções exibidas pelo comando x/i $eip estão longe de corresponder ao que você escreveu em seu programa. Elas são as instruções correspondentes à função de impressão que você evocou através da interrupção int 10h, junto com várias outras que correspondem ao trabalho normal do BIOS que independem de seu bootloader – o BIOS não deixa de ser um sistema operacional, ainda que pequeno e básico e por isso, além de rodar o seu programa, ele "presta atenção" a outras interrupções e tarefas de manutenção para as quais foi programado. Canse seus dedos um pouco e vá avançando no programa com a sequência de comandos si (gdb) e x/i $eip (QEMU), observando a janela do emulador onde, até agora, está escrito apenas Booting from Floppy... Após muito, muito tempo (pode chegar a mais de mil passos) você verá a letra "F" ser impressa em sua tela. Se você tiver tempo e paciência, verifique as interrupções que são evocadas nesse processo,

comparando-as com a lista de interrupções do BIOS encontrada nesse link: http://80864beginner.com/8086-interrupt-list/

É bom, no desenvolvimento ou depuração de sistemas operacionais, que você tenha esses e vários outros truques na manga. Mas como você quer evitar cãibras e seguir adiante em suas experiências, vamos apenas ver o programa imprimindo, um a um, os caracteres na tela. Você já descobriu que a instrução `int 10h` está armazenada na posição de memória `7C2Eh`. Assim, use o gdb para definir um ponto de parada (breakpoint) nessa posição e continue executando o programa teclando `c`, também na janela do gdb e acompanhando o que acontece na janela do emulador do QEMU.

```
(gdb) br *0x7c2e
Ponto de parada 1 at 0x7c2e
(gdb) c
Continuando.

Breakpoint 1, 0x00007c2e in ?? ()
(gdb) c
Continuando.

Breakpoint 1, 0x00007c2e in ?? ()
```

Você pode querer, também, colocar um ponto de parada na posição `7C29H` (você lembra qual instrução está nessa posição de memória) e usar uma combinação dos comandos `c` e `si` no gdb e `x /i $eip`, `info registers` e outras no QEMU apenas para acompanhar o que acontece dentro desta porção de seu programa:

```
.repeat:
   lodsb              ; Pega um caracter da frase
   cmp al, 0
   je .done           ; Se o caracter for 0, termina
   int 10h            ; Caso contrario, imprima
jmp .repeat
```

Um sistema operacional em Assembly

Sistemas operacionais modernos costumam ser escritos em linguagens de programação de alto nível, justamente para garantir a sua portabilidade entre várias plataformas de hardware. Um sistema operacional escrito na linguagem C poderá rodar em qualquer arquitetura que tenha um compilador dessa linguagem.

Não custa lembrar, mais uma vez que, independente da linguagem na qual um sistema operacional é escrito, o código que o processador executará é o assembly, a linguagem de máquina na qual o compilador transformou o sistema operacional escrito na linguagem C.

O MikeOS é um sistema operacional cuja intenção original é ser uma ferramenta de aprendizagem. Com o passar do tempo, o sistema desenvolvido por Mike Saunders passou a ter uma série de contribuições e hoje conta com interpretadores para as linguagens BASIC e FORTH, além da possibilidade da execução de programas na linguagem C. Agora que você já conhece um pouco de Assembly, que tal visitar a página do projeto e experimentar o MikeOS: http://mikeos.sourceforge.net

Navegando pela página você encontrará o link para baixar um arquivo compactado contendo o código-fonte amplamente comentado (em inglês), juntamente com várias imagens do sistema, prontas para serem carregadas pelo VirtualBox ou QEMU.

Para começar a experimentar, descompacte o MikeOS em uma pasta apropriada e, localize, no sistema já descompactado, a pasta disk_images. Nela você encontrará a imagem de disquete mikeos.flp. Coloque o sistema em operação com o comando:

```
qemu-system-i386 -fda mikeos.flp
```

Certifique-se de especificar o caminho apropriado, caso você não esteja executando o QEMU dentro da mesma pasta onde está a imagem de disquete. Imediatamente você deve ver uma tela com a mensagem:

```
Thanks for trying out MikeOS!
Please select an inteface: OK for the program menu, Cancel for command line.
```

Na pasta sources, dentro do local onde você descompactou o MikeOS, você encontra o arquivo kernel.asm. Se você procurar, dentro dele, por “Thanks”, logo irá identificar as seguintes linhas:

```
    jmp option_screen             ; Offer menu/CLI choice after CLI has ex ited

    ; Data for the above code...

    os_init_msg           db 'Welcome to MikeOS', 0
    os_version_msg        db 'Version ', MIKEOS_VER, 0

    dialog_string_1       db 'Thanks for trying out MikeOS!', 0
    dialog_string_2       db 'Please select an interface: OK for the', 0
    dialog_string_3       db 'program menu, Cancel for command line.', 0
```

Ou seja, você já pode, por exemplo, começar a traduzir o MikeOS para o português!

Na janela do QEMU, tecle OK e tome um tempo para familiarizar-se com o MikeOS.

Agora que você já brincou um pouco com o sistema, use a seta para baixo e encontre o programa MEMEDIT.BAS, como mostra a figura 4.

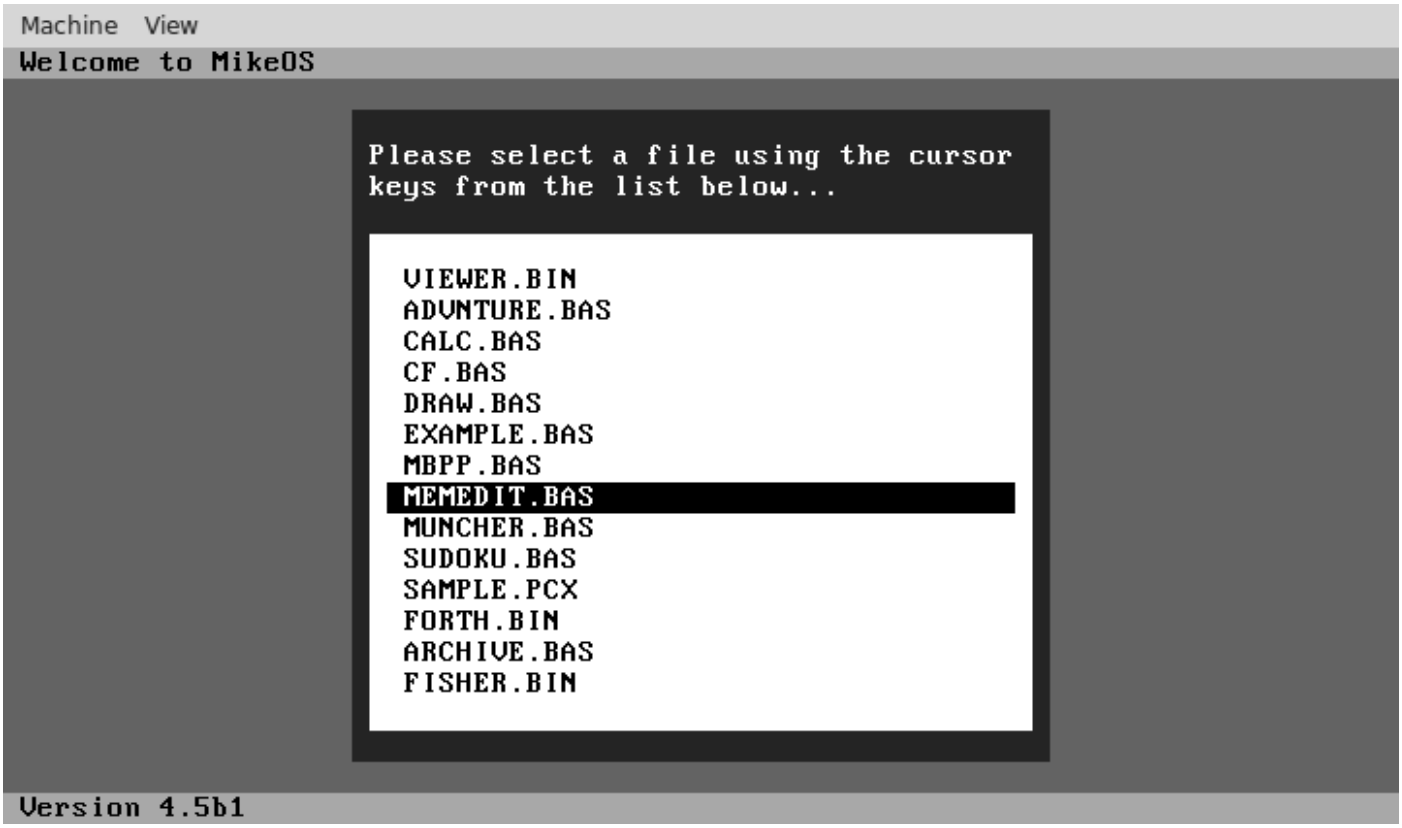

Figura 4 – Evocando o MEMEDIT.BAS

Você abrirá o *Memory Manipulator* (Manipulador de Memória), um programa muito similar ao hte que você já utilizou anteriormente. Repare que na parte inferior da tela há algumas instruções sobre como usar o programa. Tecle G (GO TO – ir para) e o número 160, como mostra a figura 5.

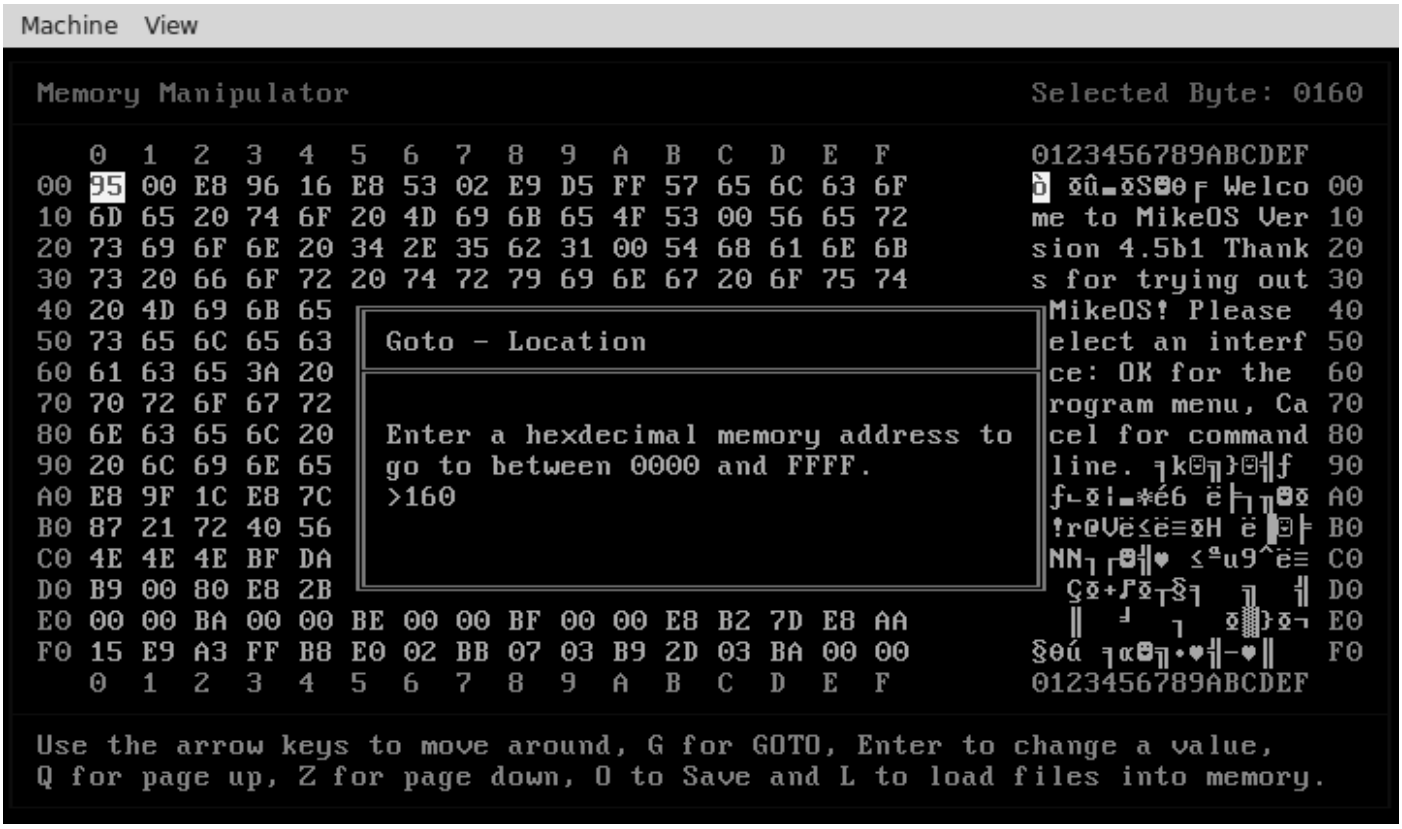

Figura 5 – Usando o Memory Manipulator

Com cuidado, substitua a expressão “Welcome to MikeOS” por “Aprendendo MikeOS”. Você terá que usar uma tabela ASCII! Posicione o cursor em cada letra que quiser mudar e use o novo código ASCII correspondente. Compare com a figura 6 e tome muito cuidado para não mudar nada que não seja, de fato, um caractere. De outra forma, você poderá estar bagunçando o próprio sistema operacional – ainda assim, não se preocupe muito com isso! Lembre que você está em uma máquina virtual QEMU. Se algo der errado, basta começar tudo de novo.

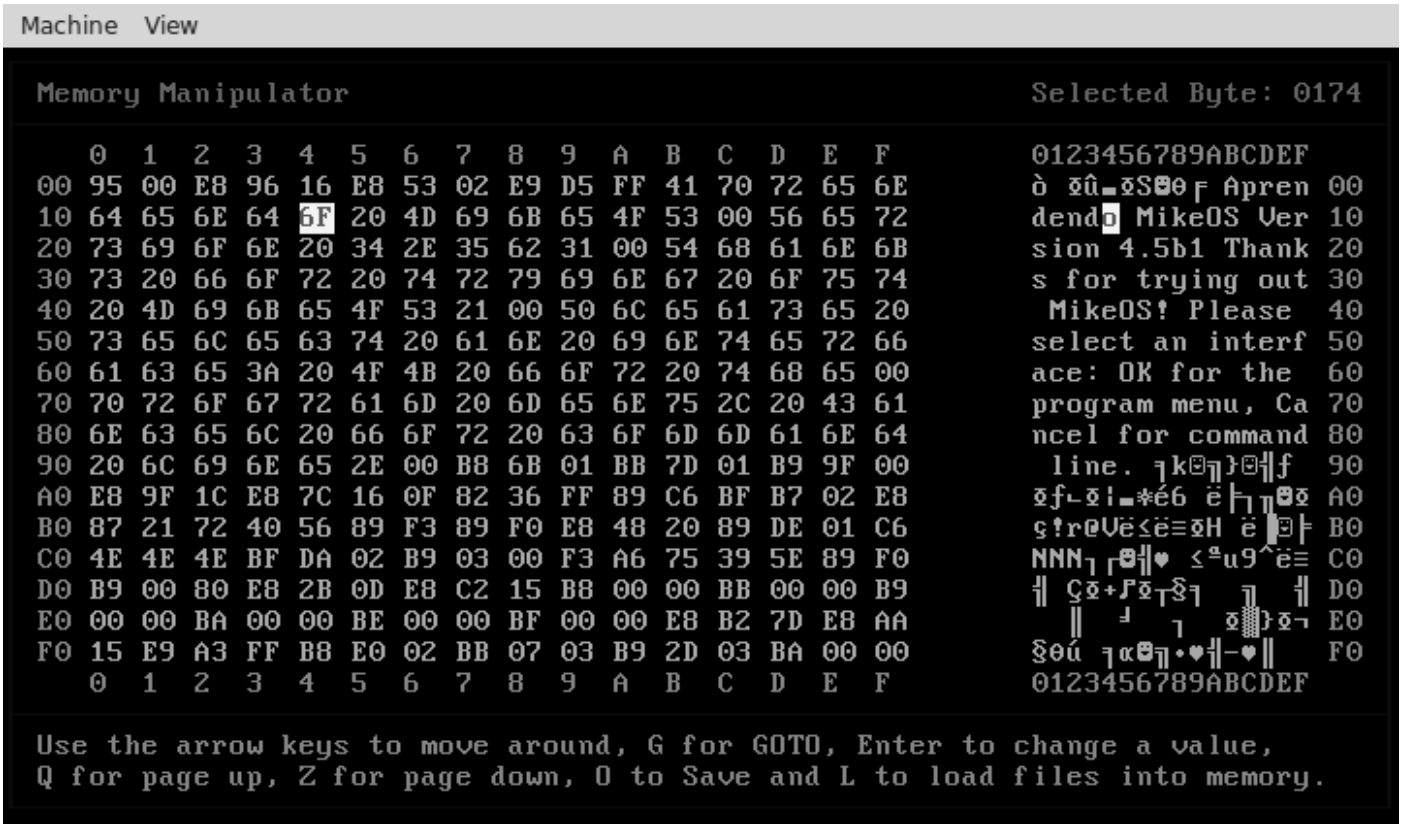

Figura 6 – Alterando posições de memórias

Uma vez completas as suas alterações, tecle ESC para sair do programa e pressione uma tecla qualquer para voltar ao menu inicial. Agora, como mostra a figura 7, no topo da tela do MikeOS você deve ver “Aprendendo MikeOS”.

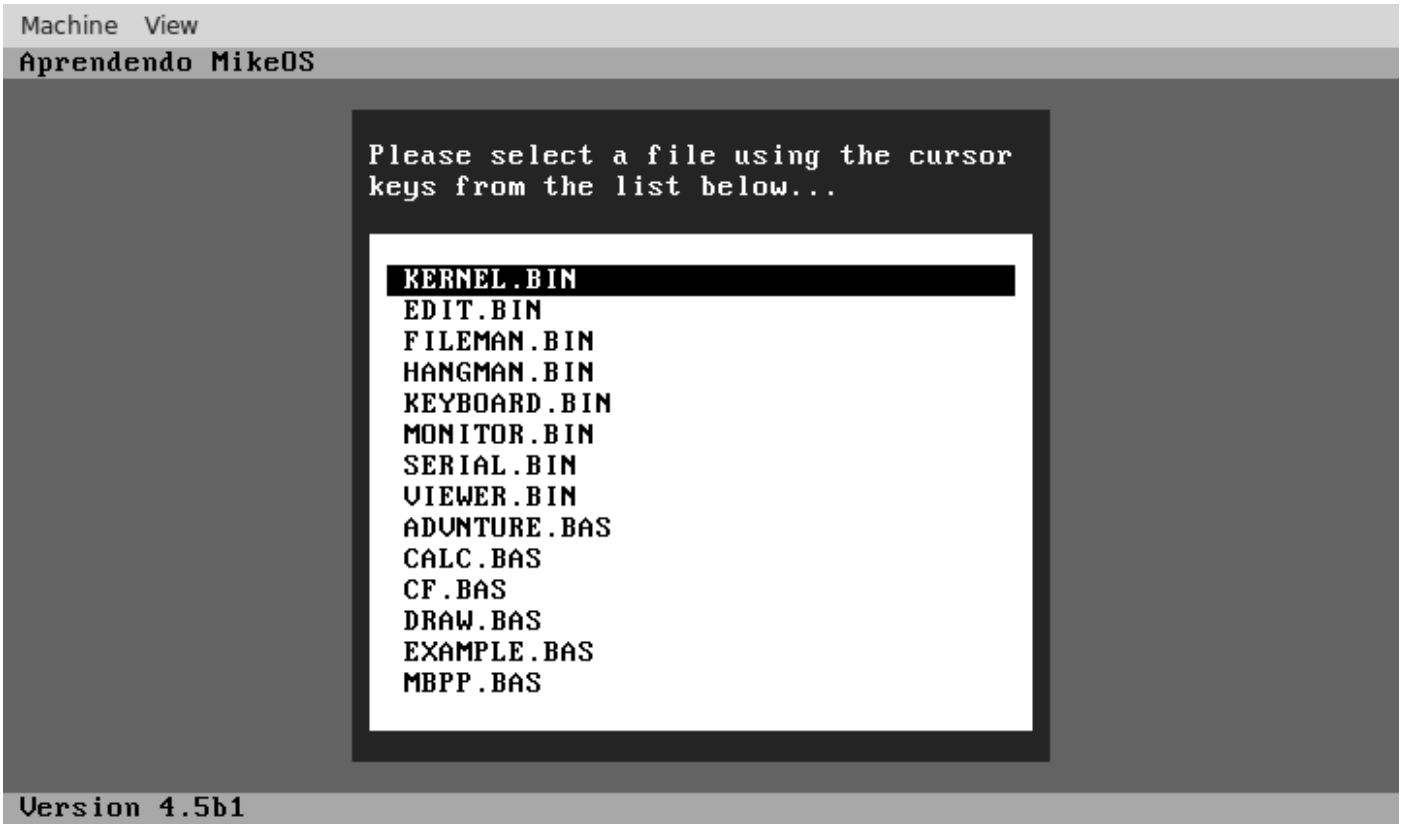

Figura 7 – Aprendendo MikeOS

Tipicamente, você não deve fazer alterações diretamente na memória de um sistema operacional em execução. Por outro lado, é importante a familiaridade com esse tipo de procedimento uma vez que ele pode ajudar bastante na depuração de códigos. Idealmente, as mudanças devem ser feitas nos arquivos fonte (no caso do MikeOS, nos arquivos .asm) e o nasm deve ser usado para a geração dos programas em código binário.

Repare, na pasta onde você descompactou o MikeOS, a estrutura de diretórios:

```
~/mikeos $ ls -la | grep ^d
drwxr-xr-x  6 brod brod  4096 Mai  8 20:44 .
drwxr-xr-x 78 brod brod 36864 Jul 11 09:30 ..
drwxr-xr-x  2 brod brod  4096 Mai  8 20:44 disk_images
drwxr-xr-x  2 brod brod  4096 Mai  8 20:40 doc
drwxr-xr-x  2 brod brod  4096 Mai  8 20:38 programs
drwxr-xr-x  4 brod brod  4096 Mai  8 20:08 source
```

O comando ls lista os arquivos de uma pasta, a chave -la faz com que a listagem seja feita de maneira detalhada e o comando grep ^d faz com que apenas as linhas que começam com a letra d, indicando um diretório, sejam exibidas. Se você se interessou em seguir adiante seu aprendizado em Assembly e no MikeOS, a pasta doc contém manuais para o desenvolvimento de aplicações e modificações para o sistema. A pasta source contém o código fonte do sistema operacional e a pasta programs o código fonte de todos os programas acessórios, incluídos no MikeOS. O manipulador de memória que você acabou de usar tem seu código em mikeos/programs/memedit.bas.

O suprassumo do bagaço do resumo

Há alguma chance de que essa seja a sua primeira leitura sobre sistemas operacionais e você deve estar com os olhos inchados ao final desse texto. Espero que eles estejam inchados pela curiosidade e não por este livro estar fazendo você chorar. O importante é você saber que sua máquina só entende zeros e uns e que a representação mais próxima deles, para humanos, é a linguagem Assembly e os códigos ASCII. Você pode escrever programas em qualquer linguagem de mais alto nível mas, no final da contas, eles serão traduzidos, por compiladores, nesses zeros e uns.

Nas experiências que você já fez e fará no restante desse livro você usará máquinas virtuais, que são programas que simulam máquinas reais (como o VirtualBox e o QEMU). Afinal, você não quer correr o risco de corromper o sistema operacional e seus arquivos mas quer ter toda a chance de errar e aprender com os seus erros e acertos. Depuradores, como o gdb, permitem que você acompanhe o que está acontecendo dentro de uma máquina, inspecionando sua memória e seus registradores.

Leitura complementar

TANENBAUM, Andrew S. *Sistemas Operacionais Modernos*. São Paulo: Prentice Hall, 2010.

TANENBAUM, Andrew S.; WOODHULL, Albert S. *Sistemas Operacionais – Projeto e Implementação*. Porto Alegre: Bookman, 2002.

MOTA, João Eriberto Filho. *Descobrindo o Linux*. São Paulo: Novatec, 2012.

HYDE, Randall. *Art of Assembly Language*. Estados Unidos: No Starch, 2010.

BROD, Cesar; KÄFER, Joice; et al. *Redes sem Fio no Mundo em Desenvolvimento*. Estados Unidos: Hacker Friendly, 2008 (disponível gratuitamente no endereço http://wndw.net).

SCRODER, Carla. *Redes Linux - Livro de Receitas*. Jacaré: Alta Books, 2009.