Aula 13

Carregador de boot e o kernel

13.1 Carregadores de boot

Basicamente, o que vimos na aula 11, foi a inicialização do sistema. Para tanto, verificamos o processo a partir do momento em que o kernel passa a comandar o funcionamento da máquina.

Neste trecho, o que vamos discutir é um dos passos que compõe o processo de inicialização do sistema: O carregador de sistema operacional. Esta aplicação é a primeira; aquela invocada pelo BIOS, que em seguida carrega uma imagem de kernel que supostamente está armazenada em alguma unidade de disco já reconhecida.

Para muitos, o carregador de boot não tem muita importância, ou muitas vezes passa despercebido, mas, no entanto, para administradores de servidores e rede, o carregador de boot acaba sendo um software importante cuja familiaridade e conhecimento são necessários para as tarefas diárias.

Chegamos ao exagero de alguns entusiastas darem mais importância ao carregador de sistemas do que ao próprio sistema em si. Uma citação retirada do manual do GNU GRUB feita por Gordon Matzigkeit ilustra bem isso:

> *"I, personally, believe that this is a grave injustice, because the boot loader is the most important software of all. I used to refer to the above systems as either "LILO" or "GRUB" systems.*
> *Unfortunately, nobody ever understood what I was talking about; now I just use the word "GNU" as a pseudonym for GRUB. So, if you ever hear people talking about their alleged "GNU" systems, remember that they are actually paying homage to the best boot loader around... GRUB! "*

Entendendo a importância do carregador do boot, vamos analisar a ferramenta mais popular atualmente encontrada nas plataformas GNU/Linux: O GRUB.

13.2 GRUB

A história do GRUB começou quando Erich Boleyn, iniciou em 1995, um projeto para modificar o carregador do FreeBSD a fim de que esse carregador seguisse as recomendações GNU para um software *multiboot*. Percebendo a dificuldade, Erich optou por construir desde o "zero" um novo projeto para carregador de S.O. . Com isso nasce o GRUB, que aos poucos vem sendo melhorado e desenvolvido com base em uma licença GNU.

Os aspectos técnicos e de manipulação deste carregador serão mostradas de maneira sucinta nos

itens que seguem.

13.3 Capacidades do GRUB

Dentre as principais características que o GRUB apresenta, podemos citar as seguintes propriedades:

- Reconhecimento de binários ELF e a.out.
- Suportar kernel que respeitam as exigências de um sistema multiboot, como os kernel 32 bit de código livre (FreeBSD, Linux, NetBSD e OpenBSD).
- Permite o carregamento de módulos.
- Interface de execução (CLI).
- Possui arquivo de configuração para módulos e comandos do carregamento do kernel.
- Suporta diversos sistemas de arquivos como BSD FS, FAT 16, FAT 32, Minix FS, ext2, Reiserfs, etc.
- Suporta boot pela rede.
- Independente da geometria do disco.
- Consegue acessar qualquer dado em dispositivos que tenham sido reconhecidos pelo BIOS.

Além dessas qualidades, é possível elencar outras mais, entretanto, é mais importante fazer a analise e compreensão de como se dá o funcionamento e configuração do mesmo.

13.4 Funcionamento

Antes de iniciarmos a explicação de como funciona o GRUB, é necessário entendermos que existem duas versões bastante distintas de GRUB, aquela conhecida como versão 1 (GRUB legacy) e a versão 2 (de fato versão 1.99).

Apesar de ainda estar em versão beta, o GRUB 2 já é o software padrão para carregador das distribuições Debian e Ubuntu. O restante das distribuições mais famosas ainda usam como base a versão 1 do GRUB como padrão. Inicialmente será mostrado o funcionamento, de maneira rápida, da versão 1 e em um segundo momento, serão apresentadas as diferenças principais da versão 2.

13.4.1 Como funciona a versão 1

Em primeiro lugar, o BIOS busca a *master boot record* de um disco rígido, ou o setor *bootável*, para acessar um programa carregador de SO. Por conta disso, o GRUB deverá ter seu programa armazenado nesta seção do disco.

O problema é que este setor *bootável* ou a MBR terá tamanho, em geral, de 512 bytes para discos rígidos. Isto porque este é o tamanho da menor unidade de um disco. O importante é entender que esse tamanho é insuficiente para conter todo o código do GRUB. Portanto, este se fragmenta em mais de um segmento. Portanto, o que estará contido no setor bootável da partição, é o programa

conhecido como *stage1*.

O *stage1* tem uma referência direta a um outro conjunto de setores do disco (os setores seguintes ao stage1) onde estará parte do programa GRUB chamado de s*tage1.5*. Esse *stage1.5* é estritamente dependente do sistema de arquivos em questão. Ele indica como deve ser mapeado o sistema de arquivo para que seja possível encontrar a imagem completa do programa GRUB. O terceiro fragmento, é então o *stage2*, que nada mais é do o núcleo do GRUB a ser invocado pelo *stage1.5*.

Quando o *stage2* sobe à memória, tem-se então o inicio do processo de boot para um sistema operacional. Agora sim este processo pode ser caracterizado de duas formas a saber:

a) Carregando um sistema direto.

Para carregar um sistema diretamente, o GRUB deve indicar qual a partição do disco é considerada a raiz. É possível realizar isso na interface de controle através do comando *root*.

O próximo passo, mais importante, é a escolha da imagem de kernel a ser descompactada na memória. O GRUB tem a capacidade de descompactar imagens em tempo de carregamento das mesmas, reduzindo o espaço utilizado em disco. Mais uma vez, para se realizar isso na interface de comando é necessário executar o comando *kernel*.

Um último comando é o *module*, que é opcional e serve para carregar alguns módulos juntamente com a imagem de kernel escolhida.

Após esses três passos, é possível finalizar o processo de carregamento do sistema, com o próprio carregamento do kernel em si. O comando final para isso é a instrução de *boot*.

b) Acionando um segundo carregador através encadeamento.

Quando o GRUB necessita carregar algum sistema operacional que não suporta carregadores *multiboot*, ele deve indicar a partição em que se encontra esse sistema e trabalhar como se fosse um BIOS, ou seja, carregando na memória o próximo carregador de SO. Exatamente por isso, esse modo é denominado de carregamento em cadeia.

A sequência de comandos em uma interface do GRUB para isso é composta pelos comandos:

- *rootnoverify* (dispositivo,partição): O GRUB vai indicar qual é o disco e a partição que deverá conter o carregador de SO.
- *makeactive*: Tornar ativa essa partição.
- *chainloader* +(setor): Quantos setores deverão ser lidos para que o carregador de SO dessa partição seja carregado na memória. No caso das plataformas microsoft tradicionais, temos “chainloader +1”, carregando um único setor.
- *boot*: carregar o SO, que de fato estará executando um novo (próximo) carregador.

13.4.2 Sintaxe e convenções

Conforme sabemos, os sistemas Linux possuem uma regra para nomes de dispositivos de entrada e saída. Essas regras não são seguidas de forma idêntica no GRUB, mas ainda assim, é fácil e intuitivo, uma vez que já se conhece as convenções para entrada e saída independente de dispositivo no Linux.

Para o GRUB, um dispositivo deve ser descrito como:

([dispositivo][número],[partição],[BSD PC slice])

A partição é um número contado a partir de 0. Já o BSD slice, necessário somente para sistemas BSD, necessário se o GRUB deve procurar por uma partição formatada para estes sistemas. Para sistemas Unix BSD, as partições (ou slices) possuem caracteres que rotulam sua natureza. Um *slice* que contém o rótulo ‘a’ deve ser uma partição *bootável*.

13.4.3 Arquivos de configuração

Por padrão, a configuração do GRUB fica dentro do diretório /*boot* em um subdiretório que leva o próprio nome */boot/grub*.

Além da configuração, nesse diretório, encontramos as imagens do GRUB: *stage1*, *reiserfs_stage1_5* e *stage2*.

Os arquivos de configuração mais importantes são o *menu.lst e o device.map*

- *menu.lst:* contém as configurações que o GRUB deve ler. Neste arquivo é onde estão as determinações a serem seguidas pelo GRUB quando este é executado.

A sintaxe é bastante simples. Contendo um conjunto de opções para cada uma das imagens de kernel. Cada opção é iniciada através de um título. Após o título da opção de imagem, seguem os comandos que determinam a sequência necessária para que o GRUB carregue um sistema (root, kernel e boot).

As linhas de um exemplo de arquivo *menu.lst* são transcritas abaixo:

```
title       Ubuntu 9.04, kernel 2.6.28-11-generic
uuid         90bb9e6a-9973-4bc7-a96f-e108e6dc74f7
kernel       /boot/vmlinuz-2.6.28-11-generic root=UUID=90bb9e6a-9973-4bc7-a96f-e108e6dc74f7 ro quiet splash
initrd      /boot/initrd.img-2.6.28-11-generic quiet

title       Ubuntu 9.04, kernel 2.6.28-11-generic (recovery mode)
uuid         90bb9e6a-9973-4bc7-a96f-e108e6dc74f7
kernel       /boot/vmlinuz-2.6.28-11-generic root=UUID=90bb9e6a-9973-4bc7-a96f-e108e6dc74f7 ro  single
initrd      /boot/initrd.img-2.6.28-11-generic

title       Ubuntu 9.04, memtest86+
uuid         90bb9e6a-9973-4bc7-a96f-e108e6dc74f7
kernel       /boot/memtest86+.bin quiet
```

As linhas *title* indicam as três opções de boot para o GRUB. Todas elas estão na mesma partição do mesmo disco, pois se notarmos o UUID (Universallly Unique Idntifier), veremos que este é sempre o mesmo. Esse identificador nada mais é do que um índice que é gerado para determinar exclusivamente uma partição. Os sistemas de arquivos nativos em Linux possuem suporte ao UUID, estes identificadores únicos de partições são a garantia de exclusividade na referência, mesmo quando a configuração física de discos é alterada.

O comando *kernel* indica qual é a imagem, nestes casos, os arquivos a serem buscados para cada uma das opções (vmlinuz-2.6.28-11-generic e memtest86+)

Outras opções além de *root*, são *ro*, *quiet* e *splash*. A primeira indica que o sistema de arquivos deve ser carregado como apenas leitura. Isso é feito por questão de segurança. O sistema passa a permitir escrita após que o processo *init* assim o determinar. A segunda opção indica que o log das execuções não vão para tela, fazendo o boot de forma "*quiet*". Por fim, a opção *single* indica que o *kernel* deve receber o parâmetro para *single user*. Isto é feito em caso de recuperação ou problemas. Com isso o kernel em memória não terá suporte a multiusuários.

O *initrd*, passado juntamente com cada imagem é na verdade também uma imagem que deve ser carregada em memória e que tem como principal função auxiliar temporariamente ao kernel durante o boot. Em muitos casos o *initrd* é uma imagem genérica que contém diversos módulos necessários para que o sistema seja iniciado corretamente. O *initrd* é opcional e é muito útil quando se deseja criar um boot que contemple plataformas de diferentes ambientes que exigem no tempo de boot alguns módulos pré-requisitos para o sistema base.

É possível visualizar qual módulo contém uma imagem de initrd. Basta montá-la em algum local de seu sistema de arquivos, como segue:

```
#> mount -t cramfs -o loop initrd.img /mnt
```

Continuando com as outras linhas importantes para a configuração do GRUB. Dentre as de maior destaque pode-se elencar as seguintes:

```
default 0
fallback 1
hiddenmenu
password --md5 $1$gLhU0/$aW78kHK1QfV3P2b2znUoe
timeout        3

 title        Windows 95/98/NT/2000
 root         (hd0,0)
 makeactive
 chainloader   +1

 title        Linux
 root         (hd0,1)
 kernel       /vmlinuz root=/dev/hda2 ro
```

O principal aqui é determinar o significado de *default*. Como podemos deduzir, este define o padrão de imagem a ser escolhida na inicialização. A contagem do GRUB tem inicio em 0 e termina na quantidade de titles ou opções de boot existentes dentro de *menu.lst*. Para o exemplo acima, teríamos como padrão, o boot do sistems Windows, através da cadeia feita com *chainloader*.

Quando o kernel da imagem default ou qualquer outro que tenha sido selecionado, não puder ser carregada com sucesso, o GRUB utiliza alternativamente o carregamento da opção listada em *fallback*. Neste exemplo, se a imagem 0 não for carregada com sucesso, o GRUB tenta automaticamente carregar a imagem 1.

A opção *hiddenmenu* determina que o menu interativo do GRUB deve estar oculto durante sua execução e há uma espera (*timeout*) de 3 segundos para que usuário possa interromper o prosseguimento do GRUB através da tecla *ESC*.

A opção *password* indica que para que o usuário possa utilizar o GRUB, editando ou passando comandos, ele deve fornecer a senha correta. Neste exemplo a senha esta criptografada, mas a mesma pode ser armazenada em texto puro.

13.4.4 Como funciona a versão 2

Depois de termos visto como funciona a versão 1, é possível fazermos uma análise comparativa para entendermos efetivamente como funciona a versão 2 do GRUB.

A primeira questão a se tomar nota é em relação aos arquivos e diretórios de configuração. De forma bastante prática podemos elencá-los assim:

- /etc/default/grub, contendo as definições padrões do menu do GRUB.

- /etc/grub.d/ é o diretório que contém os scripts interpretados capazes de montarem o arquivo de configuração.

- /boot/grub/grub.cfg é o arquivo de configuração utilizado pelo aplicativo GRUB como diretriz de funcionamento.

Em detalhes, um exemplo de arquivo de definições (default) pode ser visto abaixo:

```
# If you change this file, run 'update-grub' afterwards to update
# /boot/grub/grub.cfg.
GRUB_DEFAULT=0
GRUB_HIDDEN_TIMEOUT=0
GRUB_HIDDEN_TIMEOUT_QUIET=true
GRUB_TIMEOUT="10"
GRUB_DISTRIBUTOR=`lsb_release -i -s 2> /dev/null || echo Debian`
GRUB_CMDLINE_LINUX_DEFAULT="quiet splash"
GRUB_CMDLINE_LINUX=""
# Uncomment to disable graphical terminal (grub-pc only)
#GRUB_TERMINAL=console
# The resolution used on graphical terminal
# note that you can use only modes which your graphic card supports via VBE
# you can see them in real GRUB with the command `vbeinfo'
#GRUB_GFXMODE=640x480
# Uncomment if you don't want GRUB to pass "root=UUID=xxx" parameter to Linux
#GRUB_DISABLE_LINUX_UUID=true

# Uncomment to disable generation of recovery mode menu entrys
#GRUB_DISABLE_LINUX_RECOVERY="true"
```

Se formos deduzir as definições inseridas nesse arquivo, poderemos notar que basicamente são definições que modificam a apresentação e características do menu inicial do GRUB (*default*, *timeout*, *command line default*, etc).

Por sua vez o diretório */etc/grub.d/* é aquele responsável em armazenar os arquivos de imagem do aplicativo e scripts de configuração que ao serem interpretados pelo GRUB podem montar o arquivo final de configuração. De verdade, o arquivo final de configuração, *grub.cfg*, é gerado quando se executa o binário *update-grub*. Tal aplicativo interpreta os shells script do diretório grub.d e ao final escreve o resultado no arquivo /boot/grub/grub.cfg. Por esse motivo, se recomenda fortemente que não se faça uma edição direta do arquivo grub.cfg como costumávamos realizar na versão 1. Toda modificação agora deve ser realizada dentro do diretório do grub.d seguido de uma execução imediata do update-grub.

A listagem de arquivos exemplificando o diretório de scripts do grub pode ser vista assim:

```
-rwxr-xr-x   1 root root  3296 2009-10-23 22:44 00_header
-rwxr-xr-x   1 root root  1154 2009-10-23 22:31 05_debian_theme
-rwxr-xr-x   1 root root  3778 2009-10-23 22:44 10_linux
-rwxr-xr-x   1 root root   772 2009-10-23 14:11 20_memtest86+
-rwxr-xr-x   1 root root  5467 2009-12-07 21:08 30_os-prober
-rwxr-xr-x   1 root root   214 2009-10-23 22:44 40_custom
-rw-r--r--   1 root root   483 2009-10-23 22:44 README
```

Aqui, visualizamos os arquivos, que agora representam as entradas de menu, conforme a antiga configuração de GRUB que era possível se encontrar no arquivo *menu.lst.* Por isso, cada um dos arquivos deverá conter os comandos GRUB capazes de descreverem as definições para uma opção de menu. A título de exemplificação um novo arquivo para um sistema chamado “windows 8” poderia ser visto assim:

```
#!/bin/sh -e
echo "Adding Windows 8 to GRUB 2 menu"
cat << EOF
menuentry "Windows 8" {
set root=(hd0,1)
chainloader (hd0,1)+1
}
EOF
```

Evidentemente, o arquivo para essa configuração não contém somente instruções e operações nativas do GRUB, mas sim, um arquivo de shell script que delimita as instruções de GRUB através de “cat << EOF” e “EOF”.

Se notarmos bem, as instruções entre os delimitadores (redirecionamento de entrada) são similares à versão 1 do GRUB, então a tarefa de criar imagens novas (personalizadas) não será algo tão traumático para usuários antigos do GRUB.

Outra importante citação a se fazer é a sequência com que os arquivos são listados dentro do diretório. Isto é, nesse exemplo, o arquivo do windows 8 poderia receber o nome de *15_windows8* e assim ser colocado entre as imagens *10_linux e 20_memtest86+*. Isso é importante para saber qual será o número atribuído para essa nova entrada de imagem dentro do menu do GRUB. O conceito é o mesmo utilizado para se ordenar os scripts de inicialização no modelo do Sys init V.

Além de não dispor mais de um arquivo único, como o menu.lst, o GRUB 2, agora denomina as partições através de números iniciados em 1 e não mais em 0 como se utilizava na versão anterior. Isto é, o que na versão 1 utilizaríamos (hd0,0), hoje denotaríamos por (hd0,1).

Avaliando esses arquivos e definições, pode-se notar que há uma diferença enorme entre as versões. Isto porque, a versão 2 do GRUB, foi na verdade, uma implementação de software totalmente reescrita com pouco aproveitamento de código. A partir disso somos obrigados a entender como estão dispostos os binários e o passo de execução do GRUB em sua versão mais atual.

13.4.4 Os estágios da versão 2

Diferente da versão 1, o que vemos na versão 2 são dois estágios bem definidos. Agora não há mais um arquivo binário denominado de stage2, mas sim, um diretório onde existem módulos e imagens de binários necessários para o funcionamento total do GRUB.

Analisando o diretório /boot/grub/ podemos ver diversos arquivos. A maioria deles possuem nomes com prefixos .mod e .img . Isso significa que são arquivos binários e de módulos, onde o

principal arquivo é o core.img.

Esse arquivo core.img pode ser interpretado como uma imagem, tal qual uma initrd, mas nesse caso é como se fora uma imagem minimizada de kernel que por sua vez deve carregar módulos e objetos para que posteriormente possam “enxergar” todas partições com imagens de sistema operacional adequadamente. A imagem de “core” é quase como se fosse um sistema operacional, guardadas proporções e, naturalmente, com uma finalidade bastante específica.

Portanto, comparando a versão 1 com a versão 2, podemos dizer que o arquivo a ser carregado inicialmente é o boot.img que por sua vez aciona o arquivo core.img. Após essa imagem core estar na memória, os módulos necessários são chamados, dependendo da demanda de cada um deles.

Exemplos de arquivos de imagens são: jfs.mod, linux.mod, hdparm.mod, raid5.mod, xfs.mod, etc.

O restante do funcionamento não difere em nada da versão 1. Isso pode ser dito pois ainda é possível ter o carregamento direto ou o carregamento por cadeia, além do que, é possível integrar as versões de sistemas encadeados com versões de GRUB 1 com GRUB 2.

Conhecendo, agora, os carregadores de kernel, vamos analisar melhor como ter a possibilidade de personalizar e criar uma nova imagem de kernel para ser carregado em um sistema GNU/Linux.

13.5 O Kernel

O código fonte do kernel do Linux é robusto e flexível o suficiente, permitindo a compilação do mesmo de forma que tenhamos imagens modulares ou monolíticas. Ambas são seguras e estáveis viabilizando seu uso em diversos ambientes diferenciados.

A característica principal do kernel do Linux, naturalmente herda a intenção do projeto inicial de seu idealizador, Linux Torwalds, que na tentativa de construir um sistema seguro e eficaz, propôs um kernel monolítico .

Mas como em todo projeto, a necessidade de mudança e as exigências de adequações rápidas de código, levaram ao kernel do Linux, a necessidade de suportar módulos. Isto fez com que ele fosse classificado como um Kernel modular ou híbrido.

Ao se compilar o kernel do Linux é possível selecionar as opções: embutido, modular ou não incluso. Dessa forma, pode-se construir um kernel com as rotinas e objetos todos embutidos, determinando-o como monolítico. Por outro lado, um kernel apresentando funções e rotinas modulares mais as embutidas, torna-se um kernel modular.

Há grandes discussões acerca dos tipos de kernel e quanto a sua eficiência e confiabilidade. Entretanto, apesar de muito interessante, este não é o escopo do texto e pode ser mais detalhado em literaturas teóricas a respeito do assunto.

13.6 Passando parâmetros ao kernel durante o boot

Como fora dito na seção do GRUB, sabe-se que é possível passar algumas variáveis em tempo de carregamento para o kernel. Estas variáveis certamente tem a função de modificar o carregamento

ou a execução padrão de um kernel em memória.

Logo abaixo estarão listados algumas dessas variáveis ou parâmetros. Os 10 mais utilizados, e suas definições são:

- *single*: Quando passada ao kernel, em tempo de carregamento, indica que o mesmo deve ser carregado para operar em modo monousuário. Isto serve para recuperação em caso de desastres ou problemas.

- *root*: Parâmetro que informa qual a raiz do sistema de arquivos. Deve receber o valor de uma partição. Esse exemplo já fora mostrado no GRUB

- *ro*: Determina que partição raiz deve se montada em modo apenas de leitura. Por padrão, sempre o kernel solicita a montagem como partição somente de leitura. Posteriormente, o processo init é quem muda essa característica.

- *rw*: Determina que a partição raiz seja carregado com permissão de leitura e escrita.

-*panic*: Quando habilitada, esse parâmetro deve informar quantos segundos esperar para que seja reiniciado o sistema em caso de panic. Por exemplo, o sistema pode reiniciar após 10 segundos com *panic=10.*

- *maxcpus*: Informa o número máximo de CPUs que um kernel deve utilizar quando carregado. por exemplo *maxcpus=2*.

- *debug*: Quando ativado, esse parâmetro exige a exibição de informações de debug. Útil para desenvolvedores que estejam interessados nas mensagens internas.

-*raid*: Quando a rotina para *RAID*s *md* estiver compilada de forma embutida no kernel, é possível informar ao kernel qual será o dispositivo de array de raids. Por exemplo informando que deve ser o dispositivo 2, *raid=/dev/md2*.

-*selinux*: Ativa ou desativa o modo de kernel projetado pela NSA (National Security Agency). Quando ativado (selinux 1), faz com que o kernel seja carregado em modo *Security-Enhanced Linux* (selinux). O padrão é estar desativado (selinux 0).

-*mem*: Informa quanto de memória do sistema deve ter endereçado. Serve para casos de testes e nos casos em que o kernel não consegue ou não deve enxergar toda memória do sistema.

13.7 O System.map e sua serventia

Variáveis e funções, para nós humanos, são inteligíveis através de convenções de linguagem como por exemplo: *fopen( ), BytesRead( ), ip_forward*, etc.

Entretanto, sabemos a máquina referencia os endereços, que, de fato serão os dados binários contidos nesses endereços. Portanto, quando o kernel em execução retorna algum erro, ele simplesmente estará em algum endereço ou referência de memória, do ponto de vista da máquina.

A grande confusão se dá quando o humano tenta entender qual erro ocorreu através do recebimento do endereço ou referência como base.

Neste cenário é que entra o arquivo de mapeamento chamado de *System.map*. Este é um arquivo automaticamente gerado em toda compilação do kernel que serve para correlacionar os endereços das variáveis e funções do kernel (símbolos) em endereços de memória.

Com isso, ao executar um instrução qualquer, o sistema de log do kernel (*klogd*) poderá recebê-la e traduzi-la em um símbolo mais inteligível do que um simples endereço. Esse mapeamento auxilia e muito ao administrador na busca e entendimento de erros e problemas de execução do kernel.

Um exemplo de trecho de um arquivo de mapeamento, *System.map* pode ser:

```
c0105930 T do_coprocessor_segment_overrun
c01059b0 T do_invalid_op
c0105a50 T do_bounds
c0105ad0 T do_overflow
c0105b50 T do_divide_error
c0105bf0 T math_emulate
c0105c40 T math_state_restore
c0105cf0 T restart_nmi
c0105d10 T stop_nmi
c0105d30 T math_error
c0105ec0 T do_coprocessor_error
c0105ef0 T do_simd_coprocessor_error
c0106120 T arch_irq_stat_cpu
c0106190 T arch_irq_stat
c01061c0 t show_other_interrupts
```

Onde, à esquerda, temos o endereço da instrução no kernel e mais à direita, temos o símbolo do kernel em si.

É com base nessa tabela, que o sistema de eventos do kernel, pode, nos casos de tradução estática (existem módulos que fazem tradução dinâmica de símbolos), enviar ao sistema qual é a váriavel ou função vinculada ao endereço que esta sendo referenciado no momento.

Essas referencias são bastante úteis nos casos de erros como os conhecidos *oops* ou *panic*. Há saber, o *oops* é um erro gerado em módulo ou rotina que permite a continuidade das próximas instruções ou módulos, abortando apenas o escopo local desse *oops*. Já o *panic*, consiste em um congelamento do sistema, exigindo seu novo carregamento,

13.8 Compilando o Kernel

Como já vimos em textos anteriores, existe a possibilidade de se alterar o kernel em que o Linux utiliza, dado que este tem o código fonte aberto.

Dessa forma, é possível recorrer às últimas versões de Kernel, disponíveis no repositório oficial (*http://www.kernel.org*) e através desse código fonte, compilar uma versão mais personalizada de sistema.

Como já sabemos, para se compilar um código um fonte, podemos utilizar a facilidade das ferramentas *make*. E neste caso, não será diferente.

O passo a passo, que podem ser encontrado em centenas de ajudas de compilação do kernel ao longo da Internet, basicamente envolve os comandos primordiais. Mais especificamente, pode-se listá-lo na ordem de execução:

- Baixar o código fonte

- Descompactá-lo em */usr/src* através de: *tar -xjvf linux.tar.bz*

- Uma vez descompactado, pode-se realizar o *configure*. Como a compilação exige um passo de configuração com muitas variáveis. O *configure* aqui não é idêntico ao que estamos familiarizados. O arquivo *config* gerado, no qual o *make* deve receber como entrada, pode ser editado manualmente ou então ser editado através de uma ferramenta mais "amigável" e interativa. Para tanto, o *make* invoca o *menuconfig* que dá a possibilidade da criação do arquivo *.config* através de um prompt de interações:

#> make menuconfig.

- Uma vez, escolhidas as opções de módulos ou rotinas embutidas, o código estará pronto para ser compilado. Para isso, executa-se o *make*, exigindo a geração da imagem compactada do kernel através de: *make bzImage*

- Caso, a escolha durante a etapa de configuração tenha sido por um kernel modular, é necessário compilar os módulos escolhidos. Este passo requer a compilação e a instalação destes módulos através de: *make modules && make modules_install*

- Feito isso, teremos a imagem do kernel pronta no diretório *arch/x386/boot/bzImage* e o arquivo de *System.map* dentro do diretório inicial do código fonte (*/usr/src/linux-xyz*).

- O trabalho final requer em copiar a imagem *bzImage* e o *System.map* para o diretório de boot (por padrão em /*boot)*. Daí basta gerar uma imagem *initrd*, pois poderá ser necessário para alguns módulos em tempo de inicialização. O comando para gerar essa imagem é o *mkinitramfs* usando como referência o diretório base dos módulos do kernel utilizado.

#> cp arch/x86/boot/bzImage /boot/vmlinuz-2.6.x.y
#> cp System.map /boot/System.map-2.6.x.y
#> mkinitramfs -o /boot/initrd.img-2.6.x.y 2.6.x.y

- Naturalmente, após tudo isso, deve-se editar a configuração do carregador do SO. Neste caso, tratamos do GRUB. Para tanto, devemos analisar se estamos trabalhando com a versão 1 ou com a versão 2 do GRUB.

Caso seja a versão 1 do GRUB, deve-se editar o arquivo */boot/grub/menu.lst* de forma que sejam adicionadas as linhas de título, imagem com raiz e imagem de *initrd* do novo *kernel*.

Caso seja utilizada a versão 2 do GRUB, deve-se criar um novo arquivo contendo as informações do novo kernel criado no diretório */etc/grub.d/* . Alternativamente, se estivermos utilizando a versão 2 do GRUB e colocarmos os nomes das imagens seguindo a convenção de nomes utilizada nos exemplos (*vmlinuz-2.x.y e initrd.img-2.x.y*), é possível simplesmente atualizar o GRUB para obter a configuração final. Isto é, através da execução do comando *update-grub*, a nova entrada do kernel já estará disponível no arquivo de configuração */boot/grub/grub.cfg*.

- Terminado, é possível reiniciar o sistema e selecionar o novo kernel durante a execução do GRUB.

Para maiores informações, deve-se consultar as páginas de manuais do kernel que estão em constante reformulação (http://www.linuxdocs.org/HOWTOs/Kernel-HOWTO.html).

13.9 Atividade

Com base no artigo em referência (ver site), responda as seguintes questões:

1) Descreva sucintamente quais foram as motivações e necessidades para a reescrita por completo de uma nova versão de GRUB.

2) Fazendo uma comparação entre as versões 1 e 2 do GRUB, e com base no artigo, descreva quais as principais diferenças entre ambos para os seguintes items:

- sintaxe de nomenclatura para partições
- arquivos de configuração
- binários e execuções
- adição de nova imagem e edição do menu inicial

3) Explique, como você poderia, através de comandos do GRUB , iniciar um procedimento para recuperar a senha de administrador do sistema. Em contrapartida, explique um outro procedimento, também no GRUB, capaz de mantê-lo protegido, evitando que esse tipo de técnica venha a ser utilizada por pessoas não autorizadas.

4) Descreva o passo a passo para que seja obtida a última versão estável do kernel, e com a mesma, seja compilado um novo kernel com suporte à NETFILTER - ARP TABLES. Explique quais são os passos para compilação e inclusão da nova imagem do kernel no carregador de SO em questão.