Stefan Hellwig - www.sxc.hu

Primeira aula da preparação LPIC-2

LPI nível 2: Aula 1

Conheça o kernel Linux e saiba como aplicar patches, configurá-lo, compilá-lo e manipular seus módulos.

por Luciano Siqueira

The Linux Professional Institute

Tópico 201: O kernel Linux

2.201.1 Componentes do kernel

O candidato deve ser capaz de utilizar os componentes do kernel que são necessários para hardware, drivers, recursos de sistema e requerimentos específicos. Este objetivo inclui saber implementar diferentes tipos de imagem do kernel, identificar kernels estáveis, em desenvolvimento e patches, além de saber usar os módulos do kernel.

O kernel é o componente central do sistema. É tão importante que muitas vezes é confundido com o sistema em sua totalidade. Ou seja, apesar de *Linux* designar apenas o componente central – o kernel – o termo é normalmente utilizado para designar todo o sistema, que é composto de muitos outros programas. Por isso, muitos desenvolvedores e figuras importantes do mundo do Software Livre preferem nomear o sistema *GNU/Linux*, dado que a maior parte dos programas que funcionam em conjunto com o kernel Linux fazem parte do projeto GNU, cujo propósito é manter um ambiente de desenvolvimento e ferramentas o mais próximo possível de suas contrapartes do Unix, porém obedecendo ao modelo de desenvolvimento aberto.

Portanto, o kernel é o componente central do sistema, responsável pela comunicação com o hardware, gerenciamento de processos, entre outros.

O kernel e seus módulos

Existem basicamente duas formas de se escrever um kernel para qualquer plataforma: um kernel monolítico ou um micro-kernel. O kernel Linux é monolítico. Diferente de um micro-kernel, um kernel monolítico agrega todas as funções dentro de um único processo. Já um micro-kernel delega cada função específica a processos derivados.

Porém, o kernel Linux trabalha modularmente. Isso significa que, mesmo sendo um kernel monolítico, todas as suas funções não precisam necessariamente estar presentes na memória. Por exemplo, o kernel pode estar configurado para trabalhar com dispositivos USB, mas não manter em memória as funções exigidas para trabalhar com os mesmos. Mantidas em módulos, essas funções somente serão carregadas para a memória quando forem necessárias, ou seja, quando for conectado um dispositivo USB.

É importante não confundir um kernel modular com um micro-kernel. Apesar de modular, o kernel Linux é um kernel monolítico. Cada módulo carregado é integrado ao kernel ativo e, ape-

sar de em sua maioria poderem ser descarregados da memória, o kernel continua se comportando como único e centralizado.

Versões do kernel

A versão de um kernel Linux é instituída através de quatro números. Esses quatro números juntos informam não só a versão do kernel, mas também se trata-se de um kernel estável ou em fase de testes e desenvolvimento.

Dada a natureza colaborativa do desenvolvimento do kernel, novas funções e correções são incluídas num ritmo diário. Porém, nem todas as novas funções foram suficientemente testadas para serem consideradas prontas para uso em ambientes de produção em geral. Por esse motivo, existem sempre duas versões de kernel mantidas: uma chamada "estável" e outra "de desenvolvimento".

O formato do número de versão do kernel é *A.B.C.D*. O último elemento (*D*) nem sempre é utilizado, mas tem função muito importante.

O que representa cada elemento na versão do kernel:

- **A:** Versão propriamente dita. Muda apenas quando ocorrem transformações radicais na estrutura do kernel. Está atualmente na versão 2.
- **B:** Número de revisão principal. Até a versão 2.4 do kernel, esse segmento, quando par, significava que tratava-se de uma versão estável. Quando ímpar, tratava-se de uma versão em desenvolvimento, e portanto considerada instável. Da versão 2.6 em diante, instituiu-se não utilizar mais essa distinção entre par e ímpar.
- **C:** Número de revisão secundário. Antigamente, determinava correções e patches de segurança. Hoje demonstra se houve inclusões de novos recursos, como drivers de dispositivos, por exemplo.
- **D:** Utilizado quando há revisões de segurança urgentes, mas que não produzem grande alteração no código.

Existem ainda alguns sufixos normalmente utilizados em kernels extra-oficiais, como *rc1* ou *smp*. Esses sufixos demonstram alguma particularidade do kernel em questão. No caso citado, *rc1* significa *release candidate 1* e *smp* significa *Symmetric Multiprocessing*, capacidade de aproveitar todos os recursos de máquinas com mais de um processador central ou com ao menos um processador de múltiplos núcleos.

Patches do kernel são trechos de código contendo correções. Como na maioria das vezes em que uma correção é necessária apenas uma pequena parte do código do kernel precisa ser alterada, somente esse trecho é distribuído para ser aplicado ao código previamente distribuído.

Exemplos de nomes usados para distribuir o kernel Linux:

`linux-2.4.22.tar.gz`: Código-fonte completo do kernel 2.4.22.

`patch-2.6.21.1.bz2`: Primeiro patch corretivo para a versão 2.6.21 do kernel.

Arquivos

Por padrão, todo o código-fonte do kernel é mantido na máquina local dentro do diretório `/usr/src/linux`. Ali encontram-se não só os arquivos de código-fonte do kernel, mas também a documentação oficial e onde estará o arquivo imagem do kernel após compilado.

A documentação oficial está em */usr/src/linux/Documentation*. Neste diretório encontram-se vários arquivos de texto que documentam aspectos específicos do kernel. Por exemplo, para descobrir quais parâmetros o kernel pode aceitar, o arquivo *kernel-parameters.txt* pode ser consultado.

Após compilar um novo kernel a partir de seu código-fonte, o arquivo imagem será encontrado em `/usr/src/linux/arch/i386/boot/`. Note que o subdiretório `i386` varia conforme a arquitetura escolhida durante a configuração do kernel. Se foi escolhida a arquitetura *PowerPC*, por exemplo, a imagem estará em `/usr/src/linux/arch/powerpc/boot/`.

O nome e o tamanho da imagem variam de acordo com o método de compilação utilizado. A imagem do kernel se chamará `zImage` ou `bzImage`. O nome *zImage* demonstra que a imagem está comprimida com o método *zlib*. O nome *bzImage* (*big* zImage) demonstra que a imagem também foi comprimida com o método *zlib*, porém não é limitada ao espaço restrito de memória obrigatório em máquinas antigas. Não confundir bzImage com o método de compressão *bzip2*.

2.201.2 Compilando um kernel

Apesar de a maioria das distribuições acompanharem kernels pré-compilados, pode ser necessário personalizar o kernel para corresponder a necessidades específicas, como suporte a hardware incomum ou um sistema de arquivos exótico. O código-fonte do kernel pode ser obtido através de pacote específico da distribuição ou diretamente do site oficial `www.kernel.org`. No último caso, o código é distribuído como um *tarball*, e deve ser extraido no local padrão, `/usr/src/linux`, que geralmente é um link simbólico para o diretório criado na extração do tarball, `/usr/src/linux-x.x.xx`.

O processo de personalização de um kernel exige três etapas principais: configuração, compilação e, por fim, instalação.

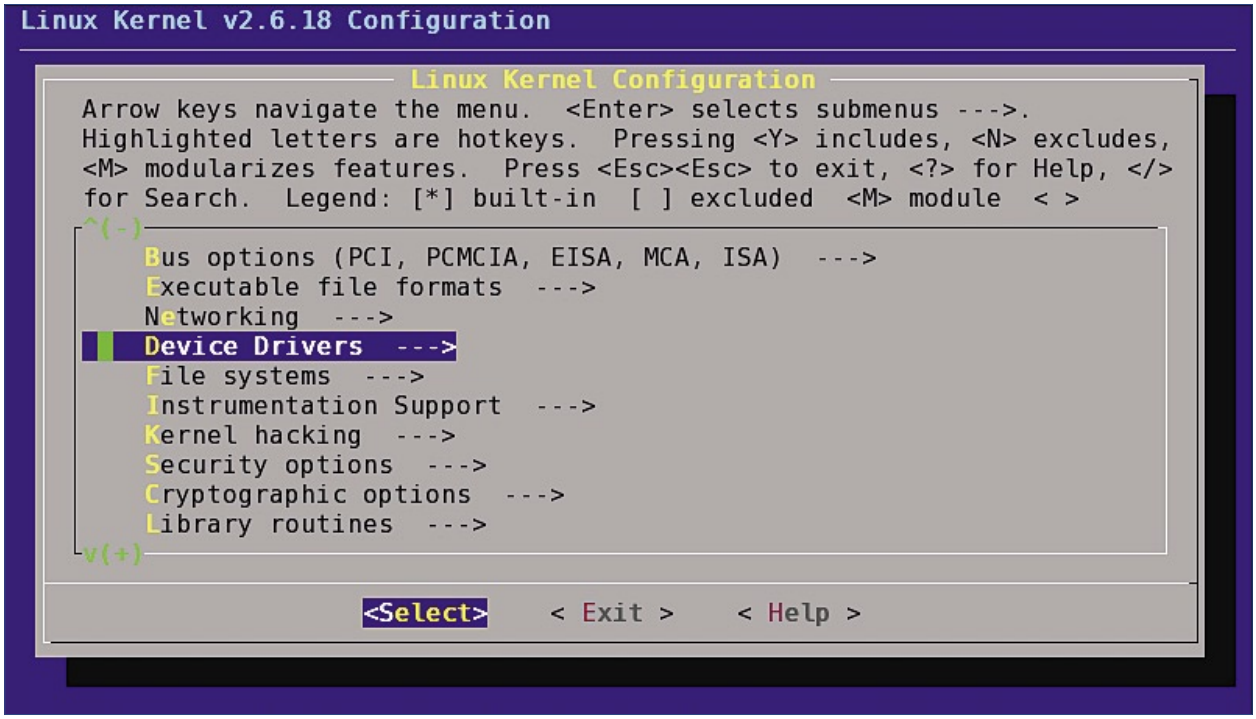

Figura 1 Configuração do kernel pela interface *ncurses*.

Configuração

A configuração de um novo kernel é feita invocando-se o utilitário *make*. Existem diferentes tipos de interfaces de configuração, mas a mais tradicional é a interface *ncurses*, invocada através do comando `make menuconfig` (**figura 1**).

Outras formas de configurar o kernel são:

- `make config`: Criar ou atualizar a configuração através de uma interface orientada a perguntas e respostas na linha de comando;
- `make xconfig`: Configurar através de uma interface gráfica baseada em *QT*;
- `make gconfig`: Configurar através de uma interface gráfica baseada em *GTK*;
- `make oldconfig`: Atualizar a configuração utilizando um arquivo `.config` como base;
- `make cloneconfig`: Criar uma cópia das configurações do kernel atual.

Enquanto que alguns recursos precisam ser compilados como estáticos (embutidos no kernel), a maioria pode ser compilada como dinâmico (módulo carregável e descarregável). O item marcado com asterisco (`*`) será compilado como estático, e aqueles marcados com a letra *M* serão compilados como módulos. Itens deixados em branco não serão compilados. Espaços de escolha `[ ]` indicam que o item só poderá ser compilado como estático, enquanto espaços de escolha `< >` indicam que o item poderá ser compilado tanto como estático quanto como módulo.

A configuração do kernel está dividida nos eixos principais:

- *Code maturity level options*: Mostrar ou não recursos considerados instáveis;
- *General setup*: Características gerais do kernel. É possível incluir um termo de versão para o kernel personalizado;
- *Loadable module support*: Suporte ao sistema de módulos e definição de algumas características;
- *Processor type and features*: Indica o tipo de processador que o kernel utilizará, e recursos como multiprocessamento;
- *Power management options (ACPI, APM)*: Opções relativas ao controle de energia. Indicado especialmente para laptops;
- *Bus options (PCI, PCMCIA, EISA, MCA, ISA)*: Suporte aos diferentes tipos de barramentos;
- *Executable file formats*: Tipos de arquivos que o sistema será capaz de executar;
- *Networking*: Suporte e configuração dos diferentes tipos de plataformas de rede;
- *Device Drivers*: Escolha e configuração dos dispositivos de hardware, integrados e periféricos;
- *File systems*: Lista de sistemas de arquivos compatíveis e recursos relacionados;
- *Kernel hacking*: Opções de depuração do kernel.

Essas são as principais categorias de configuração, e podem diferenciar de uma versão do kernel para outra.

As configurações são salvas no arquivo `/usr/src/linux/.config`, que será usado para guiar a construção do novo kernel e módulos. No ar-

quivo `Makefile` é possível mudar variáveis como `EXTRAVERSION`, que indica ser uma compilação de kernel personalizado.

Compilação

Gerado o arquivo `.config`, o kernel está pronto para ser compilado. Se não for a primeira compilação do kernel, pode ser conveniente executar `make clean`, que apaga a maior parte dos arquivos gerados pela compilação anterior, mas mantém o arquivo de configuração. Para remover não apenas os arquivos gerados, mas também as configurações anteriores, pode ser utilizado o comando `make mrproper`. Nesse caso, será necessário criar uma nova configuração para o kernel antes de compilá-lo.

Na versão 2.4 do kernel, é necessário mais um passo antes da compilação: `make dep`. Serão geradas as dependências para que não sejam utilizadas funções desnecessárias ou conflitantes.

Para gerar a imagem `zImage` ou `bzImage` do kernel, utiliza-se `make zImage` ou `make bzImage`, respectivamente. Os comandos apenas funcionarão se for obedecida a grafia correta, com o `I` maiúsculo em `zImage` e `bzImage`.

Muito provavelmente o novo kernel será modular, e os módulos precisam ser compilados separadamente. Isso é feito através do comando `make modules`.

Desde a versão 2.6 do kernel, pode ser necessário criar uma imagem raiz inicial, chamada `initrd` (*initial ram disk*), para o correto funcionamento do sistema. O arquivo `initrd` é necessário, por exemplo, se o suporte ao sistema de arquivos utilizado na partição raiz foi compilado como módulo.

Para criar um arquivo `initrd`, é utilizado o comando `mkinitrd`. Porém, o uso desse programa é um pouco diferente para cada distribuição.

Ao arquivo `initrd` podem ser incluídos os módulos necessários à inicialização do sistema, por exemplo, os módulos para lidar com os sistemas de arquivos *ReiserFS* e *XFS*. Para criar o arquivo `initrd` com as opções padrão, basta invocar o comando `mkinitrd` sem argumentos. O comando criará o arquivo `/boot/initrd.gz` ou `/boot/initrd.img`.

Instalação

Após compilados o kernel e os módulos e criado o `initrd`, o novo kernel está pronto para ser instalado. No caso do sistema usar o carregador de boot *LILO*, basta atualizar no arquivo `/etc/lilo.conf` as informações para utilização do novo kernel:

```
  image = /usr/src/linux/arch/
➥i386/boot/bzImage
    root = /dev/sda1
    initrd = /boot/initrd.gz
    label = novo_kernel
    read-only
```

É recomendável manter o kernel anterior como opção no carregador de boot, e apenas incluir o novo kernel, para que seja possível iniciar o sistema com o kernel anterior caso haja algum problema. Feitas as alterações, o novo kernel é instalado executando o comando `lilo`.

Se for utilizado o carregador de boot *Grub*, o arquivo `/boot/grub/menu.lst` deve ser alterado. Inclua as seguintes linhas, com as opções correspondentes ao seu novo kernel:

```
title Novo kernel
  root (hd0,0)
  kernel /usr/src/linux/arch/i386/
➥boot/bzImage root=/dev/sda1 ro
  initrd /boot/initrd.gz
```

Para o Grub, não é necessário executar nenhum comando após a alteração no arquivo de configuração. O novo kernel aparecerá automaticamente no menu de inicialização.

2.201.3 Aplicando um patch no kernel

Nem sempre é necessário copiar todo o código-fonte do kernel para conseguir uma atualização ou suporte a um novo dispositivo. É possível utilizar um *patch* para que apenas os trechos relevantes de código sejam alterados.

Como o código-fonte oficial do kernel, os patches são fornecidos em arquivos *tar* compactados, por exemplo, `patch-2.6.18.1.bz2` ou `patch-2.6.18.1.bz2`.

O programa utilizado para aplicar um patch tem o nome não surpreendente de *patch*. Um patch deve ser aplicado a partir do diretório raiz do código-fonte. No caso do kernel, em `/usr/src/linux`.

Dentro do arquivo patch existem as localizações e nomes de cada arquivo que precisará ser alterado, e quais são as alterações.

A maneira mais prática de aplicar um patch é direcionar todo o conteúdo de um arquivo patch para a entrada padrão do comando patch. Isso pode ser feito numa única linha, utilizando o comando `bzcat` para arquivos `.bz2` ou `zcat` para arquivos `.gz`.

Por padrão, os patches oficiais trazem uma letra ou um termo antes do caminho completo a partir da raiz do código-fonte:

```
a/arch/i386/mm/boot_ioremap.c
```

Esse caminho indica que uma alteração deverá ser feita no arquivo `/usr/src/linux/arch/i386/mm/boot_ioremap.c`. Portanto, a letra `a` deve ser retirada do caminho, com a opção `-p1`:

```
bzcat patch-2.6.18.1.bz2 | patch -p1
```

Revertendo um patch

Agora o kernel está atualizado para a versão 2.6.18.1. Por se tratar de um patch de correção indicado pela subversão `.1`, novos patches exigirão que ele seja revertido, ou seja, que o kernel volte para o código original da versão 2.6.18, através da opção `-R`:

```
bzcat patch-2.6.18.1.bz2 | patch
➥-R -p1
```

No caso de saltos da revisão secundária, os patches são incrementais. Ou seja, não é necessário reverter um patch para aplicar um mais recente. Porém, é importante aplicar todos os patches predecessores ao desejado para que não ocorram erros.

2.201.4 Personalizando um kernel

Personalizar o kernel contempla todos os aspectos que vimos até agora: aplicar patches, configurar o kernel, compilá-lo adequadamente e instalá-lo.

Como a maioria das alterações realizadas na configuração de um kernel resultam em módulos, é importante saber manejá-los. Se foram integrados patches de terceiros, pode ser necessário editar manualmente o arquivo `.config` para editar configurações referentes aos novos recursos.

O arquivo `.config` contém linhas que correspondem a cada opção de configuração do kernel. Por exemplo, se você integrou ao kernel um patch de suporte a *ACLs*, você pode editar o `.config` nas seguintes linhas:

```
CONFIG_FS_POSIX_ACL=y
CONFIG_EXT3_FS_XATTR=y
CONFIG_EXT3_FS_XATTR_SHARING=y
CONFIG_EXT3_FS_XATTR_USER=y
CONFIG_EXT3_FS_POSIX_ACL=y
CONFIG_EXT2_FS_XATTR=y
CONFIG_EXT2_FS_XATTR_SHARING=y
CONFIG_EXT2_FS_XATTR_USER=y
CONFIG_EXT2_FS_POSIX_ACL=y
```

Ao configurar um kernel 2.2 ou mais recente, lembre-se de ativar o suporte ao *kmod*. Este é um recurso do kernel que permite o carregamento de módulos sob demanda. Quando o kernel requisitar alguma função compilada como módulo, o módulo será automaticamente carregado. Nos kernels mais antigos (2.0 e anteriores), o carregamento automático era realizado por um *daemon* chamado *kerneld*.

Considerações sobre o tópico

Este tópico não apresenta grandes novidades em relação aos conteúdos já vistos para o *LPIC-1*. Apesar disso, é esperado que o candidato conheça em detalhes os procedimentos de compilação e aplicação de patches no código do kernel. Lembre-se de quais são os principais alvos de compilação do `Makefile` do kernel, como é constituído um número de versão e de como aplicar e remover patches com o comando `patch`. ■

O autor

Luciano Siqueira é editor da Revista Easy Linux, publicada pela Linux New Media do Brasil. Formado em Psicologia, Luciano trabalha com Linux há dez anos e tem certificado LPI 1.