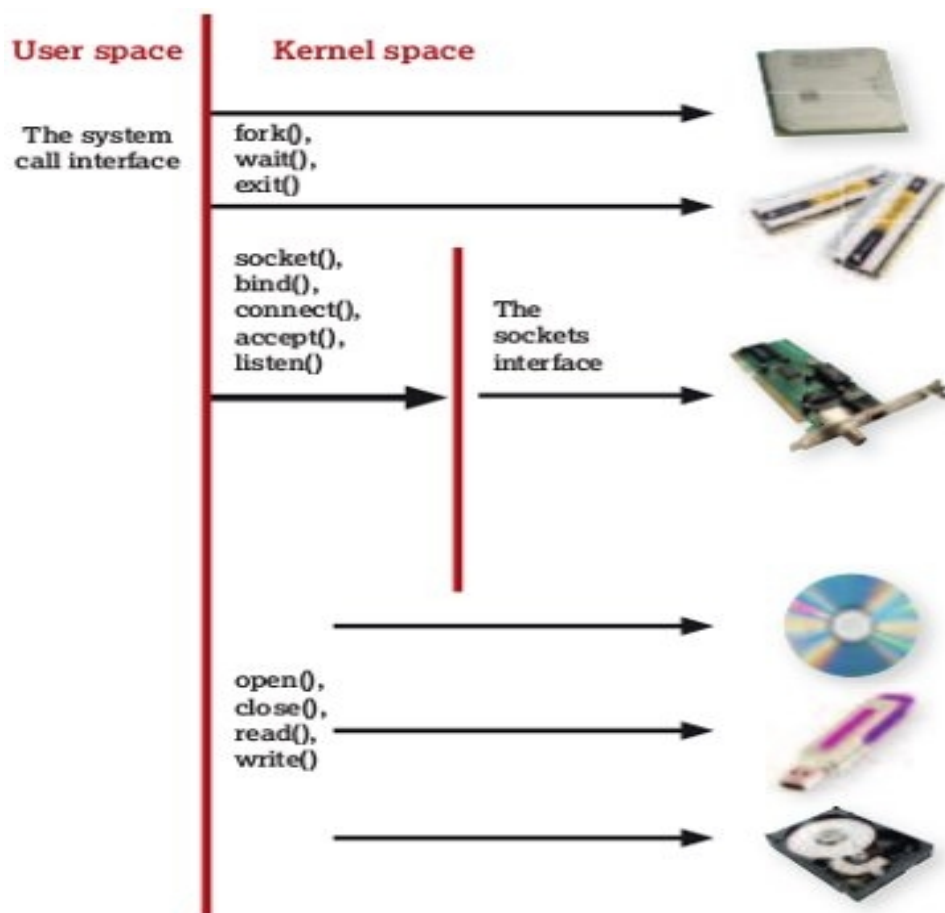


Como trabalha o Kernel Linux

Ao procurar a definição da palavra "Kernel" em um dicionário (inglês/inglês) percebi que ele deu ênfase ao seguinte ponto: *"The most important part of a statement, idea, plan, etc"* e também *"a very small part or amount of something"*. A partir daqui já temos uma idéia de que é algo muito importante, pois mesmo sendo muito pequeno o kernel é o cara que gerencia a interface de comunicação entre o hardware e os programas instalados no computador. Quando falamos do Linux estamos falando exclusivamente deste kernel, todo o resto como gnome, firefox e até mesmo o bash tratam-se apenas de programas que rodam no Linux e não fazem parte do Sistema Operacional (kernel linux).

Entendendo o kernel

OK, mas o que exatamente faz esse tal de kernel? A figura a seguir mostra de uma forma geral como o kernel disponibiliza serviço para os aplicativos rodando através de inúmeros pontos de entrada conhecidas como chamadas de sistema (system calls).



O kernel utiliza chamadas de sistema como leitura e escrita pra prover acesso ao hardware.

Do ponto de vista de um programador isso parece uma função comum, embora na realidade uma chamada de sistema envolva diferentes interruptores no modo de operação do "kernel space" para o "user space". Juntos esse conjunto de chamadas de sistema formam uma espécie de "máquina virtual" que trabalha antes do hardware real. Um exemplo claro disso é o sistema de arquivos.

Kernel Modular

Agora que entendemos melhor o que o kernel faz, vamos olhar mais atentamente a sua organização física. As primeiras versões do kernel eram monolíticas, ou seja, todos os módulos estavam compilados dentro de um único arquivo executável. O kernel das distribuições mais novas são modulares, ou seja, os módulos podem ser carregados no kernel em tempo de execução, isso faz com que o núcleo do kernel fique menor e não seja necessário reiniciar a máquina para carregar ou substituir novos módulos.

O núcleo do kernel é carregado na memória na hora do boot e é lido de um arquivo no diretório `“/boot”` na maioria das vezes este arquivo é chamado de `“vmlinuz-VERSÃO_DO_KERNEL”`. Para ver a versão do kernel corrente utilize o comando:

```
---  
#uname -r  
---
```

Os módulos ficam no diretório `“/lib/modules/VERSÃO_DO_KERNEL/”`

Gerenciando módulos.

Veremos agora alguns comandos para gerenciar os módulos do seu kernel, como por exemplo o comando para listar os módulos carregados que é o `“lsmod”`, o `lsmod` vai te mostrar uma saída semelhante a esta:

```
---  
Module                Size Used by  
vfat                   14464 0  
isofs                  36388 0  
fat                    54556 1 vfat  
nfs                    262540 0  
lockd                  67720 1 nfs  
nfs_acl                4608 1 nfs  
sunrpc                 185500 5 nfs,lockd,nfs_acl  
bridge                 55576 0  
tun                    12672 0  
usb_storage            73792 4  
libusual               19236 1 usb_storage  
---
```

Esta saída possui quatro campos divididos por nome do módulo que está carregado, o tamanho do módulo, quantas vezes ele está sendo utilizado e quais os módulos que dependem dele. Podemos carregar um módulo utilizando o comando `“modprobe”` (podemos também utilizar o comando `“insmod”` porém o `modprobe` é mais aconselhável pois ele resolve as dependências do módulo). Outro ponto muito importante na saída do comando `lsmod` é o terceiro campo que indica a quantidade de vezes que o módulo está sendo utilizado pois o linux não vai permitir a remoção de um módulo cujo o campo `used` seja diferente de zero, no exemplo acima vemos o módulo `“isofs”` (utilizado para dar suporte ao sistema de arquivos `“ISO”` que é utilizado em CDs) com o campo `used` `“0”` neste caso podemos remover o módulo com o comando `“modprobe -r isofs”`, após executar este comando o módulo `isofs` não vai aparecer mais quando executarmos o `lsmod`.

Não é muito comum carregar módulos manualmente no dia a dia, porém se você precisar carregar um módulo manualmente você pode incluir parâmetros específicos de cada módulo para carregá-lo, como no exemplo a seguir:

```
---
modprobe usb_storage delay_use:3
---
```

No exemplo supra-citado habilitamos o parâmetro “*delay_use:3*” para o módulo “*usb_storage*” que define um time-out de 3 segundos para o módulo procurar um novo device. Para saber quais as opções de cada módulo utilizamos o comando `modinfo` como no exemplo a seguir:

```
---
# modinfo usb_storage
filename:    /lib/modules/2.6.24-23-generic/kernel/drivers/usb/storage/usb-storage.ko
license:    GPL
description: USB Mass Storage driver for Linux
author:     Matthew Dharm <mdharm-usb@one-eyed-alien.net>
srcversion: 99E0EB653929DE200DF6AF9
depends:     libusual,usbcore,scsi_mod
vermagic:   2.6.24-23-generic SMP mod_unload 586
parm:      delay_use:seconds to delay before using a new device (uint)
---
```

A linha que nos interessa neste caso é a que começa com “*param*” que mostra os parâmetros aceitos pelo módulo. Caso você possua a source do kernel você também pode encontrar uma documentação muito útil em “*/usr/src/VERSÃO_DO_KERNEL/Documentation/kernel-parameters.txt*”

Sistema de arquivos /proc

O kernel também nos fornece inúmeras informações que podem ser encontradas no sistema de arquivos “*/proc*”, os arquivos no `/proc` são criados pelo kernel (alguns você pode alterar, outros não) um exemplo claro do tipo de informação que podemos encontrar no `/proc` esta to arquivo “*/proc/modules*” que mostra todos os módulos carregados no sistema (sim é semelhante ao comando `lsmod`. =D), no arquivo “*/proc/meminfo*” que mostra o status detalhado da memória do sistema e também o arquivo “*/proc/net/arp*” que mostra a tabela ARP (mesma tabela exibida pelo comando “`arp -a`”).

Beleza falei do `/proc` mas só citei arquivos que não podem ser alterados, são arquivos que somente nos fornecem algumas informações, mas o `/proc` não é só isso, um subdiretório do `/proc` muito importante é o “*sys*”, por exemplo o arquivo “*/proc/sys/net/ipv4/ip_forward*” define que o kernel irá encaminhar pacotes IP, a sintaxe é muito simples se o conteúdo deste arquivo for “0” (zero) o kernel não fará encaminhamento de pacotes IP e se o conteúdo for “1” (Um) o kernel fará encaminhamento de pacotes IP, esta opção deve ser habilitada se você for utilizar o linux como um roteador; ta mas como eu habilito isso?

Vamos primeiro ver como esta o arquivo:

```
---
# cat /proc/sys/net/ipv4/ip_forward
0
---
```

Vemos que o conteúdo do arquivo é “0” (zero), agora vamos habilitar o encaminhamento de pacotes IP no kernel.

```
---
# echo 1 > /proc/sys/net/ipv4/ip_forward
---
```

Pronto agora o encaminhamento de pacotes IP esta habilitado, fácil né? Outro comando que também pode ser utilizado para executar a mesma tarefa é o “*sysctl*” como no exemplo a seguir:

```
---  
#sysctl net.ipv4.ip_forward  
net.ipv4.ip_forward = 0  
---
```

No exemplo acima utilizamos o comando *sysctl* para ver o valor do *ip_forward*, para alterar este valor utilizamos o comando *sysctl* com as seguintes opções:

```
---  
#sysctl -w net.ipv4.ip_forward = 1  
---
```

Porém tem um pequeno problema aqui, se reiniciarmos a maquina este arquivo voltara a ter conteúdo igual a “0” (zero). E como solucionamos isso? A solução e muito fácil utilizamos o arquivo “*/etc/sysctl.conf*” para definir os valores que serão configurados na hora do boot, então basta acrescentarmos a linha “*net.ipv4.ip_forward = 1*” no arquivo *sysctl.conf*.

Melhorando a performance.

Muitos dos parâmetros do */proc/sys* que permitem escrita podem ser utilizados para melhorar a performance do Linux, apesar de que a configuração padrão já habilite opções que trabalham muito bem. Um exemplo de como podemos modificar o kernel para melhorar a performance de acordo com o tipo de aplicação que você vai utilizar esta no guia de instalação do Oracle 10g (<http://www.oracle.com/technology/obe/obe10gdb/install/linuxpreinst/linuxpreinst.htm>) que pede pra você configurar alguns parâmetros como o “*kernel.shmmax=2147483648*” que define o tamanho máximo de segmento de memória partilhada para 2GB. (Memória partilhada é um mecanismo de comunicação entre processos que permite que um segmento de memória de ser visível dentro do espaço de endereçamento de múltiplos processos.)

Outra modificação que podemos fazer e definir que nossa máquina não vai responder por broadcast de *icmp* (o bom e velho *ping -b 255.255.255.255*) configurando o *sysctl* da seguinte forma:

```
---  
sysctl -w net.ipv4.icmp_echo_ignore_broadcasts=1  
---
```

Agora fica a seguinte dúvida, pra ver os valores eu vou ter que dar *cat* de arquivo em arquivo ou saber todos os caminhos do comando *sysctl*? Claro que não, basta utilizar o comando

```
---  
sysctl -a  
---
```

..que ele vai exibir todos os parâmetros e seu valores porém ele não fala pra que serve cada um deles porém isso podemos encontrar os detalhes no bom e velho “*man proc*” Ta vendo, o kernel não é aquele bicho de sete cabeças que muitos imaginam....rsrsr

By CleBeer...

clebeer@gmail.com

clebeerpub.blogspot.com