Desenvolvimento do Kernel Linux

Versões 2.6.X

http://www.tchelinux.org

Douglas Schilling Landgraf
Email: dougsland@tchelinux.org

Kernel Linux

- Sobre o palestrante
- Sobre a palestra

Dicas Iniciais:

- Kernel não é algo místico :)
- Máquina exclusiva para desenvolvimento (VM pode?)
- Precisa saber inglês ?
- Desenvolvimento não é igual ao “dia-dia”
- Crashs repentinos ?
- Kernel, uma coleção de rotinas ?
- Cliente de email

Kernel Linux

Licença:

- GPL (versão 2)

  http://www.gnu.org/copyleft/gpl.html

Como funciona?

- Podemos baixar o software e alterar, desde que publiquemos este software com as licenças originais, incluindo a disponibilização do código fonte.

Como ajudar ?

- Você é desenvolvedor ?
- Você NÃO tem medo de crashs ?
- Você sabe inglês ?
- Gosta de traduzir ?

Onde começar ? TODO List ?

http://www.kernelnewbies.org
http://br.kernelnewbies.org/

http://kerneljanitors.org
http://kernelnewbies.org/KernelJanitors/Todo
http://kernelnewbies.org/KernelMentors

http://www.kernel.org

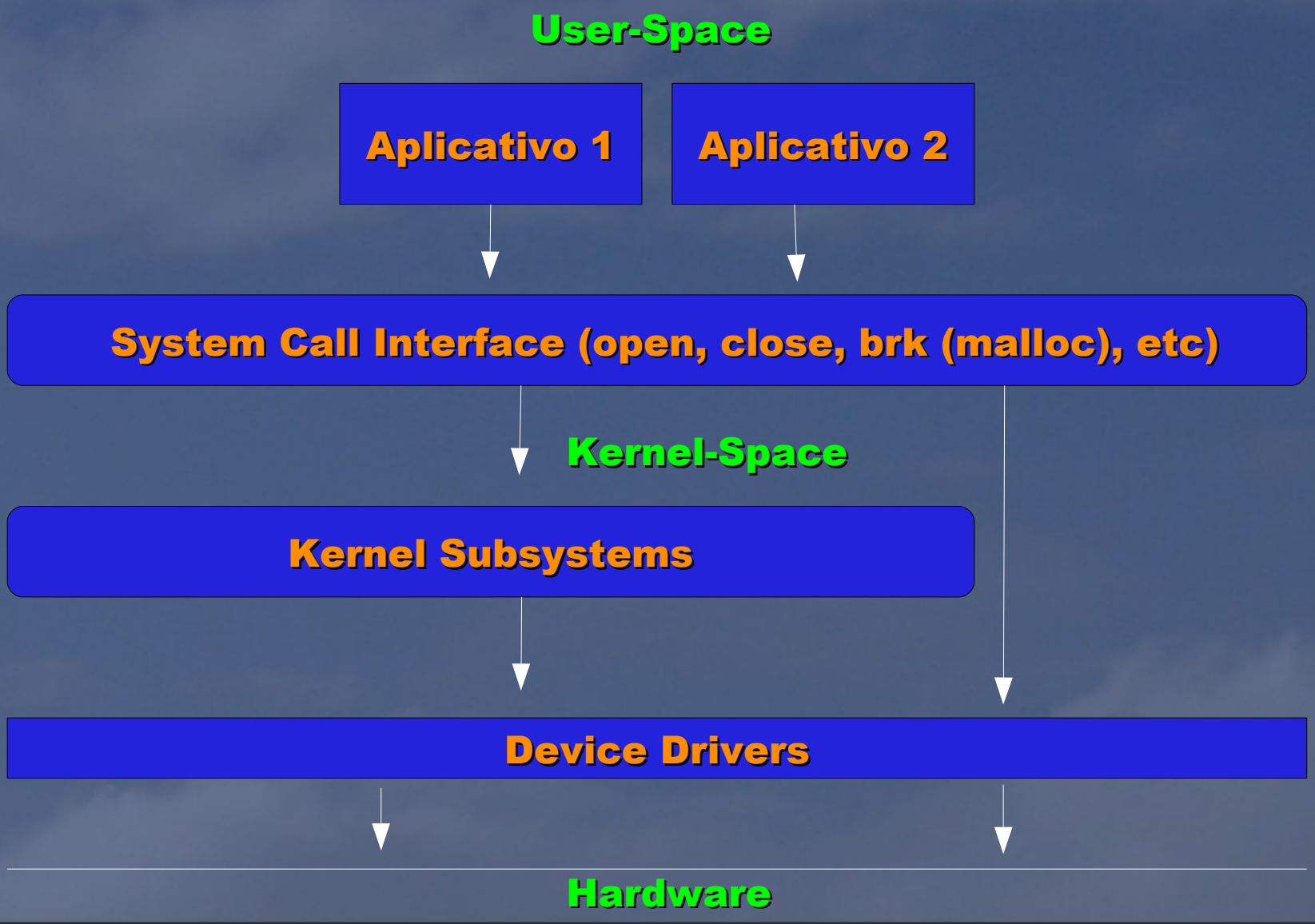
Kernel Linux

Definição:

- É no kernel que estão definidas funções para operação com periféricos (mouse, discos, impressoras, interface serial/interface paralela, usb etc.), gerenciamento de memória, entre outros.

- Coleção de rotinas que oferecem, recursos para programas tradicionais.

Kernel Linux



Kernel Linux

Onde obter o código fonte ?

http://www.kernel.org
ftp://ftp.kernel.org/pub
rsync://rsync.kernel.org/pub/

Alguns dados:

~= 9.000 arquivos C
~= 10.000 headers
~= 800 arquivos assembly

Kernel Linux

Diretórios:

arch/	Código específico de arquitetura (i386, mk68..)
block/	Dispositivos de bloco (e.x: HDs)
crypto/	API de criptografia
Documentation/	Documentação
drivers/	Device Drivers
fs/	File systems
include/	Headers
init/	Kernel boot
ipc/	Interprocess communication
kernel/	Core do Kernel

Kernel Linux

Diretórios:

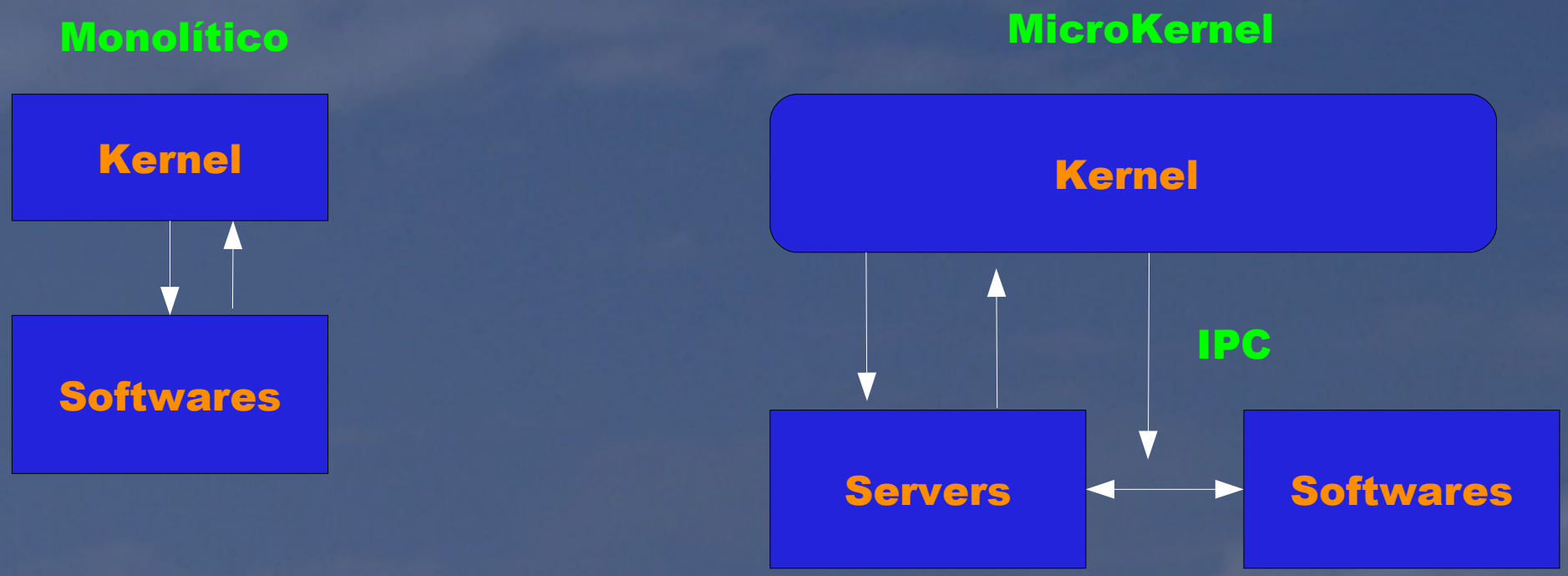
lib/	Bibliotecas
mm/	Gerenciamento de Memória
net/	Sistema de Rede
scripts/	Scripts de configuração
security/	Subsistema de segurança (SELinux)
sound/	Susbsistema de som (ALSA/OSS)
usr/	initramfs

Design: Monolítico versus MicroKernel

- Monolítico:
  - Criado em meados de 1980
  - Um grande e único processo (imenso)
  - Comunicação trivial (todos rodam em um único processo)
- MicroKernel:
  - Separado em dois processos (“servers” / user-space)
  - Comunicação via IPC (interprocess communication)
  - Modularidade

Kernel

Exemplo:



IPC = Interprocess Communication

Linux Design:

- Executa em um único processo
- Kernel Linux é monolítico
- Implementa recursos do microkernel design
- Capacidade de carregar dinamicamente módulos
- Suporte a processadores SMP (symmetrical multiprocessor)
- Sistema de acesso ao dispositivos (sysfs)

Kernel Linux

Versões:

Estáveis:

- Correções de bugs
- Novos drivers

Desenvolvimento:

- Alterações frequentes e drásticas
- Desenvolvedores testam novas soluções

Kernel Linux

Versões:

2.6.0

Versão principal é 2

Versão secundária é 6 (estável)

Release é 0

No passado como eram as versões ?

Números pares = versões estáveis
Ex.: (2.0, 2.4)

Números ímpares = versões de desenvolvimento
Ex.: (1.3, 2.3)

Ex.: linux-2.6.20.3.tar.bz2

Versões:

E hoje como funciona ?

- Hoje só temos a versão 2.6.X
- Decisão tomada no Linux Kernel Developers Summit (2004)
- Todas novas implementações são enviadas para o Andrew Morton (Árvore -mm)
- Após tornar-se estável o patch entra na versão principal.
- Versão 2.7 ~= Árvore -mm

CodingStyle:

Identação	1 TAB = 8 caracteres (!= 8 espaços)
Colunas	80

```
if (teste) {
        blah();
} else {
        bleh();
}

if (foo)
        bar();
```

$ linux/Documentation/CodingStyle

CodingStyle:

Funções:

1, 2 páginas?
< 10 variáveis locais ?

Comentários:

```
/*
 *  Olá, Eu sou um comentário!
 */
```

Outros: typedef, structs, etc.

CodingStyle:

Ferramenta indent:

$ indent -kr -i8 -ts8 -sob -180 -ss -bs -ps1 <arquivo>

ou

$ scripts/Lindent

Kbuild (compilação):

Kbuild é um framework para escrever Makefiles simples para tarefas complicadas. :)

- Economiza nosso tempo
- Comportamento similar em todas as plataformas suportadas
- Internamente ele se basea em complexos templates
- Atualmente poucas pessoas realmente sabem como ele funciona

Kbuild (exemplo):

Módulos externos

```
make -C <path> M=`pwd`
```

Módulos (Kernel que esta em execução)

```
make -C /lib/modules/`uname -r`/build M=`pwd`
```

Compilando/Carregando/Listando/Descarregando:

Atenção aos WARNINGS

$ linux/drivers/net> vi hello.c
$ linux/drivers/net> vi Makefile

$ insmod ./hello.ko
$ modprobe hello.ko (procura por dependências)

$ lsmod (lista os módulos carregados em memória)

$ modinfo ./hello.ko
$ modprobe -r hello
$ rmmod hello.ko

hello.c

```
#include <linux/init.h>
#include <linux/module.h>

static int __init hello_init(void)
{
    printk(KERN_ALERT "hello!\n");
    return 0;
}
```

hello.c

Onde esta o main() ? :)

```
static void __exit hello_exit(void)
{
    printk(KERN_ALERT "Goodbye\n");
}

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Douglas Landgaf <dougslan@tchelinux.org>");
MODULE_DESCRIPTION("Um modulo hello world!");

module_init(hello_init);
module_exit(hello_exit);
```

hello.c

E para adicionar parâmetros ?

```
module_param(io, int, 0);
MODULE_PARM_DESC(io, "Informa o io do modulo! ");

$ insmod ./hello io=0x300
```

hello.c

Exportando Símbolos

O kernel tem centenas de funções e estruturas globais
Responsável: /boot/System.map-versao-kernel

/proc/kallsyms

Module.symvers – informa os símbolos visiveis

```
shell> cat /lib/modules/`uname -r`/build/Module.symvers | wc -l
```

```
EXPORT_SYMBOL(funcao);
EXPORT_SYMBOL_GPL(funcao);
```

hello.c

Como adicionar uma entrada no /proc ?

create_proc_read_entry()
remove_proc_entry()

hello.c

Como adicionar uma entrada no /dev e /sys ?

KERNEL ="hello" executa o resto das regras quando aparecer hello /sys

SYMLINK += (append) adicionar hello_world na lista de símbolos que devem ser criados

MODE = 0444 – permite (dono, group e outros lerem)

Interfaces com o Kernel

Externas:

- System Calls
- Interface entre (userspace) e o Kernel
- Solicitações de acesso via system calls
- /proc e /sys

Internas:

- Símbolos exportados pelos módulos (EXPORT_SYMBOL)
- Ex.: printk – chamadas pública

System Calls (trace)

- Faz o trace das chamadas de sistemas (system calls)
- ptrace()

$ strace ls
$ strace -o saida.txt kradio

(programa) -> (open) GLIBC /lib/libc.so.6 -> (sys_open)

Sistema de log - Timestamps

Kernel hacking

[*] Show timing information on printks

Magic SysRq Keys

Tecla SysRq pode atribuir funções durante a execução ou travamento.

1. Alt + SysRq + R – takes the keyboard out of raw mode.
2. Alt + SysRq + E – terminates all processes (except init).
3. Alt + SysRq + I – kills all processes (except init).
4. Alt + SysRq + S – synchronizes the disk.
5. Alt + SysRq + U – remounts all filesystems read-only.
6. Alt + SysRq + B – reboots the machine.

Sistema de log - printk():

Sistema de log via /proc/kmsg
Klog -> /proc/kmsg -> syslogd

printk() ~= printf()

Sistema de log - printk():

```
printk(KERN_WARNING “mensagem de warning!\n”);
printk(KERN_DEBUG “mensagem de debug!\n”);
printk(<1> “mensagem de alerta!!\n”);
```

<linux/kernel.h> valores das MACROS {0,1,2,3, ... 7}

Prioridade: 0 -> 7

Kernel Linux

Ferramentas:

diff	Ferramenta para comparar arquivos
patch	Ferramenta para aplicar patchs
quilt	Scripts para manutenção de patchs
vimdiff	Feramenta para comparar arquivos
qemu	Emulador
git	Controle de fontes/versões
hg	Mercurial
ctags	Tags no código fonte
cscope	Navega no código fonte
ketchup	Ferramenta para atualização do kernel

git / gitk:

Autor: Linus Torvalds.
Mantenedor: Junio Hamano

Clonando árvore do Linus:
$ git clone

Manter árvore atualizada:
$ git pull

Fazer um Commit:
$ git commit

Log:
$ git log

$ git init-db / $git add . / git commit / git remove / git diff
$ git branch teste / $git checkout teste

Ctags:

```
$ make tags

$ vi .vimrc
      set tags=/usr/src/linux/tags

:ta printk
$ vim -t printk
CTRL + ]
CTRL + t
```

http://ctags.sourceforge.net.

Ferramentas Diff e Patch:

$ diff -ruN linux-x.y.z/ linux/ > meu-patch.diff

-r Recursivo
-u Formato compreensivo
-N Incluir arquivos novos

$ patch -p1 < ../meu-patch.diff (diretório abaixo)

-p1 Indica qual ponto da árvore ele vai aplicar o patch
linux/drivers/net/arquivo.c

Ferramentas Diff e Patch (exemplo):

```
--- linux-2.6.20.3.orig/drivers/net/ni65.c
+++ linux-2.6.20.3/drivers/net/ni65.c
@@ -295,7 +295,7 @@ static void ni65_set_performance(struct
 */
 static int ni65_open(struct net_device *dev)
 {
-        struct priv *p = (struct priv *) dev->priv;
+        struct priv *p = dev->priv;
```

Ferramenta Quilt:

```
$ mkdir patches
$ quilt new nome-do-patch.diff
$ quilt add nome_do_arquivo
$ quilt refresh
$ quilt top
$ quilt diff
$ quilt pop [-f] [-a]
$ quilt push [-f] [-a]
$ quilt remove
```

http://savannah.nongnu.org/projects/quilt/

Assinaturas em patches:

Signed-off-by: Assinatura

Signed-off-by: Douglas Landgraf <dougsland@gmail.com>

Acked-by: OK, sem problemas

Acked-by: Douglas Landgraf <dougsland@gmail.com>

Reviewed-by: Análise / Teste

Reviewed-by: Douglas Landgraf <dougsland@gmail.com>

Enviando um patch:

SEM anexos, patchs INLINE

Mensagem em TEXTO PURO (SEM HTML)

Você testou ?

Escolheu a lista certa?

O patch esta conforme o CodingStyle?

Mensagem: O que o patch faz com detalhes

Assunto: [PATCH] arquivo.c O que ele faz

Assinatura: Signed-off-by: Autor <email>

No máximo um patch por email (Depende de outro patch?)

Enviando um patch (exemplo):

To: kerneljanitors@....

Subject: [PATCH] ni65.c: cleanup not needed casts

Mensagem: Removed all unnecessary casts.
Signed-off-by: Douglas Schilling Landgraf <dougsland@gmail.com>

```
--- linux-2.6.20.3.orig/drivers/net/ni65.c
+++ linux-2.6.20.3/drivers/net/ni65.c
@@ -295,7 +295,7 @@ static void ni65_set_performance(struct
 */
 static int ni65_open(struct net_device *dev)
```

Instalando o código fonte:

Diretório padrão:

/usr/src/linux (Devemos usar esse path ?)

Descompactando:

```
$ tar xvjf linux-x-y-z.tar.bz2
    linux-x.y.z/Documentation/device-mapper/linear.txt
    linux-x.y.z/Documentation/device-mapper/snapshot.txt
```

.config - O que é ?

```
$ /proc/config.gz
$ /boot/config-2-6-x-y-z
```

Compilando:

$ make help	ajuda
$ make mrproper	Remover todos os arquivos + .config + backup
$ make config	modo texto
$ make menuconfig	modo texto (ncurses)
$ make xconfig	modo gráfico (Xwindows)
$ make gconfig	modo gráfico (GTK+)

Compilando (opções):

[*] - Habilitado (built-in) [ ] - Não esta habilitado

[M] - Habilitado (Módulo)

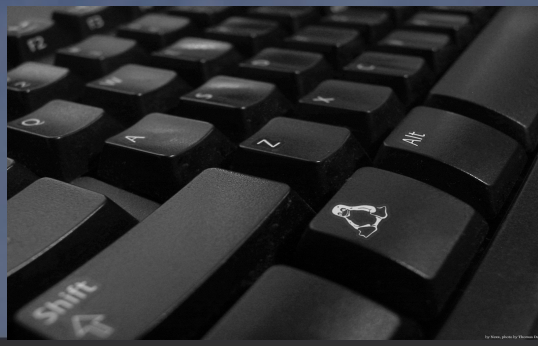
$ make Compilando o Kernel

$ make modules_install Instalando os módulos

ou

$ make modules_install INSTALL_MOD_PATH=/tmp

$ depmod -a 2.6.x.y Disponibilizar módulos para modprobe



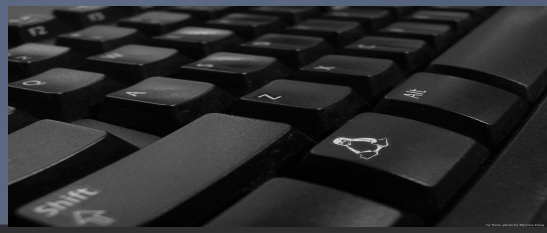
Compilando (Cross-Compilation):

Exemplo 1:

```
$> export CROSS_COMPILE=”powerpc-cross-compile-aqui”
$> ARCH=powerpc make menuconfig
$> ARCH=powerpc make
```

Exemplo 2:

```
$> make ARCH=x86_64 defconfig
$> make ARCH=arm CROSS_COMPILE=/usr/local/bin/arm-linux-
```



Kernel Linux

Compilando:

cp /usr/src/linux/arch/i386/boot/bzImage /boot/vmlinuz-2.6.X

vmlinuz = Kernel Linux Compactado

cp /usr/src/linux/.config /boot/config-2.6.X

Copiar o .config para /boot (backup)

Compilando:

cp /usr/src/linux/System.map /boot/System.map-2.6.X

System.map contém:

* Tabela de símbolos exportados (usados na depuração)
* A ferramenta top utiliza essa tabela

Compilando:

Será necessário criar uma imagem inicial para que o kernel carregue alguns módulos básicos (IDE, SCSI, RAID) antes de acessar o filesystem.

```
# mkinitrd -k /boot/vmlinuz-2.6.X -i /boot/initrd-2.6.X
```

ou

```
# mkinitrd /boot/initrd-2.6.X.img 2.6.X
```

Compilando (Grub boot loader):

$ vi /boot/grub/menu.list

```
title Kernel-2.6.X-default
root (hd0,5)
kernel /boot/vmlinuz-2.6.X-default root=/dev/hda6
vga=0x314 resume=/dev/hda5 splash=silent showopts
initrd /boot/initrd-2.6.X-default
```

Compilando (LILO boot loader):

```
$ vi /etc/lilo/lilo.conf
  image=/boot/vmlinuz-2.6.X-default
      label=2.6.X
      root=/dev/hda3
      read-only
```

$ /sbin/lilo	Grava as configurações
$ reboot	Reiniciando o sistema
$ uname -a	Exibe o kernel atual

Documentação:

linux-2.6.X/Documentation

Linux Kernel Development 2rd Edition (Robert Love)
ISBN: 0-672327201

Linux Device Drivers 3rd Edition (Cobert, Rubini, Kroah-Hartman)
ISBN: 0-596-00590-3
Versão Online (free): http://kroah.com/lkn/
Linux Weekly News
http://www.lwn.net

Google
http://www.google.com

Dúvidas ?
Sugestões?

http://tchelinux.org
http://dougsland.livejournal.com

Douglas Schilling Landgraf
Email: dougsland@tchelinux.org