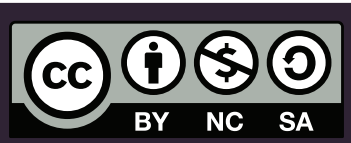
Desenvolvendo o kernel Linux: Parte II

Renê de Souza Pinto

08 de Outubro de 2013

Licença



Índice

Prática 1

- Objetivo: Apresentar o esqueleto de um módulo do kernel.
- Literatura recomendada: [Rubini e Corbet 2001]
- Passos:
  - Crie a pasta *pratica1* dentro da pasta com seu nome
  - Crie o arquivo *hellokernel.c*. Faça o esqueleto do seu módulo (declare as funções de *init* e *exit*)
  - Vamos usar a função *printk* para imprimir mensagens de debug
  - Faça o Makefile para o módulo
  - Compile e teste (insira e remova seu módulo verificando as mensagens do kernel)

Prática 2

- Objetivo: Criar um driver de dispositivo que retorna o conteúdo de um buffer, sendo acessado através de um arquivo de dispositivo.
- Passos:
  - Faça o esqueleto do módulo (pode aproveitar da prática anterior).
  - Elabore a estrutura do dispositivo e inicie-a (aloque memória se necessário, etc)
  - Aloque um *major number* para seu dispositivo (registre-o)
  - Inicie o dispositivo com as rotinas necessárias
  - Implemente as funçções open()/close() do seu dispositivo
  - Implemente a função read() do seu dispositivo
  - Tarefa: Escreva um programa em C para testar seu dispositivo

Passando parâmetros para o módulo

```
module_param(<variavel>, <tipo>, <permissao>)
```

- <variavel>: Variável que receberá o valor do parâmetro
- <tipo>: Tipo da variável
- <permissao>: Permissão para alterar/acessar o parâmetro a partir do *sysfs*

Alocando memória

```
#include <linux/slab.h>
void * kmalloc(size_t size, int flags)
```

- size: Quantidade (bytes) de memória a alocar
- flags:
  - GFP_ATOMIC: Utilizado em manipuladores de interrupções, não pode dormir.
  - GFP_USER: Usado para alocar páginas para espaço de usuário. Pode dormir.
  - GFP_KERNEL: Alocação normal para kernel. Pode dormir. Esta é a flag mais comum de ser utilizada.

Alocando dinamicamente major number

```
int alloc_chrdev_region(dev_t *dev, unsigned
  baseminor, unsigned count, const char *name);
```

- dev: Dispositivo
- baseminor: Primeiro *minor number* (geralmente é 0)
- count: Número de dispositivos que serão tratados pelo driver (ex: tty0, tty1, etc...)
- name: Nome do driver
- O *major number* atribuído pode ser visto pela diretiva MAJOR(dev) ou em */proc/devices*

Criar classe para o dispositivo

```
void class_create (struct module *owner,
  const char *name);
```

- owner: Módulo
- name: Nome da classe

Iniciar dispositivo de caractere

```
void cdev_init(struct cdev *dev,
  const struct file_operations *fops);
```

- dev: Dispositivo
- fops: Operações de arquivo (read, write, close, etc.)

Adicionar dispositivo de caractere

```
void cdev_add(struct cdev *p, dev_t dev,
  unsigned count);
```

```
void device_create(struct class *class, struct
  device *parent, dev_t devt,
    void *drvdata, const char *fmt, ...);
```

Remove dispositivo criado com *device_create*

- Remove um dispositivo criado com *device_create*.

```
void device_destroy (struct class *class, dev_t
devt);
```

Liberar major number alocado

- "Desaloca" um intervalo de número de dispositivos alocados com *alloc_chrdev_region()*

```
void unregister_chrdev_region(dev_t first,
unsigned int count):
```

Outras funções

- **kfree()**: Libera memória alocada com *kmalloc()*
- **cdev_del(struct cdev *dev)**: Finaliza dispositivo de caractere
- **class_destroy(struct class *cls)**: Remove uma classe criada com *class_create*

Considerações finais

- O ponta pé inicial foi dado. Agora é com você!
- Estude, leia, aprenda com os fontes, seja persistente
- Novos projetos, colaboração
- Sugestões, dúvidas?

Ufa! Por Hoje é só! Obrigado!

Dúvidas, sugestões, críticas?
Envie-as para rene@renesp.com.br

Referências I


RUBINI, A.; CORBET, J. *Linux device drivers*. [S.l.]: O'Reilly Media, 2001.