Projetos de Programação

Módulos de Kernel do Linux

Nesse projeto, você aprenderá a criar um módulo de kernel e a carregá-lo no kernel do Linux. O projeto pode ser realizado utilizando a máquina virtual Linux que está disponível com este texto. Embora você possa usar um editor para escrever esses programas em C, terá que usar o aplicativo ***terminal*** para compilar os programas e terá que dar entrada em comandos na linha de comandos para gerenciar os módulos no kernel.

Como você verá, a vantagem do desenvolvimento de módulos do kernel é que esse é um método relativamente fácil de interagir com o kernel, permitindo que você escreva programas que invoquem diretamente as funções do kernel. É importante ter em mente que você está, na verdade, escrevendo um ***código de kernel*** que interage diretamente com o kernel. Normalmente, isso significa que quaisquer erros no código poderiam fazer o sistema cair! No entanto, já que você usará uma máquina virtual, as falhas, na pior das hipóteses, demandarão apenas a reinicialização do sistema.

Parte I — Criando Módulos do Kernel

A primeira parte desse projeto envolve seguir uma série de passos para a criação e inserção de um módulo no kernel do Linux.

Você pode listar todos os módulos do kernel que estão correntemente carregados, dando entrada no comando

```
lsmod
```

Esse comando listará os módulos correntes do kernel em três colunas: nome, tamanho e onde o módulo está sendo usado.

O programa a seguir (chamado `simple.c` e disponível com o código-fonte para este texto) ilustra um módulo de kernel muito básico que exibe mensagens apropriadas quando o módulo é carregado e descarregado.

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

/* Essa função é chamada quando o módulo é carregado. */

int simple init(void)

{
  printk(KERN INFO “Loading Module\n”);

  return 0;
```

```
}

/* Essa função é chamada quando o módulo é removido. */
void simple exit(void)
{
  printk(KERN INFO “Removing Module\n”);
}

/* Macros para o registro dos pontos de entrada e saída do módulo. */
module init(simple init);
module exit(simple exit);

MODULE LICENSE(“GPL”);
MODULE DESCRIPTION(“Simple Module”);
MODULE AUTHOR(“SGG”);
```

A função `simple_init ( )` é o **ponto de entrada do módulo** que representa a função invocada quando o módulo é carregado no kernel. Da mesma forma, a função `simple_exit ( )` é o **ponto de saída do módulo** — a função que é chamada quando o módulo é removido do kernel.

A função de ponto de entrada do módulo deve retornar um valor inteiro, com 0 representando sucesso e qualquer outro valor representando falha. A função de ponto de saída do módulo retorna `void`. Nem o ponto de entrada do módulo nem o ponto de saída do módulo recebem quaisquer parâmetros. As duas macrofunções a seguir são usadas para registrar os pontos de entrada e saída do módulo no kernel:

```
module_init()
```

```
module_exit()
```

Observe como as duas funções de ponto de entrada e saída do módulo fazem chamadas à função `printk ( )`. Essa função é a equivalente no kernel a `printf ( )`, mas sua saída é enviada a um buffer de log do kernel cujo conteúdo pode ser lido pelo comando `dmesg`. Uma diferença entre `printf ( )` e `printk ( )` é que `printk ( )` nos permite especificar um flag de prioridade cujos valores são fornecidos no arquivo de inclusão `<linux/printk.h>`. Nessa instância, a prioridade é `KERN_INFO` que é definida como uma mensagem ***informativa***.

As linhas finais — `MODULE_LICENSE ( )`, `MODULE_DESCRIPTION ( )` e `MODULE_AUTHOR ( )` — representam detalhes relacionados com a licença do software, a descrição do módulo e o autor. Para nossos objetivos, não dependemos dessas informações, mas as incluímos porque é prática-padrão no desenvolvimento de módulos do kernel.

O módulo de kernel `simple.c` é compilado com o uso do `Makefile` que acompanha o código-fonte com esse projeto. Para compilar o módulo, dê entrada no seguinte comando na linha de comando:

```
make
```

A compilação produz vários arquivos. O arquivo `simple.ko` representa o módulo de kernel compilado. O passo seguinte ilustra a inserção desse módulo no kernel do Linux.

Carregando e Removendo Módulos do Kernel

Os módulos do kernel são carregados com o uso do comando `insmod` que é executado como descrito a seguir:

```
sudo insmod simple.ko
```

Para verificar se o módulo foi carregado, dê entrada no comando `lsmod` e procure pelo módulo `simple`. Lembre-se de que o ponto de entrada do módulo é invocado quando o módulo é inserido no kernel. Para verificar o conteúdo dessa mensagem no buffer de log do kernel, dê entrada no comando

```
dmesg
```

Você deve ver a mensagem “`Loading Module`”.

A remoção do módulo do kernel envolve a invocação do comando `rmmod` (observe que o sufixo `.ko` é desnecessário):

```
sudo rmmod simple
```

Certifique-se de fazer a verificação com o comando `dmesg` para se assegurar de que o módulo foi removido.

Já que o buffer de log do kernel pode ficar cheio rapidamente, é bom limpar o buffer periodicamente. Isso pode ser resolvido da seguinte forma:

```
sudo dmesg -c
```

Parte I — Tarefa

Execute os passos descritos acima para criar o módulo do kernel e para carregar e descarregar o módulo. Certifique-se de verificar o conteúdo do buffer de log do kernel usando `dmesg` para garantir que você seguiu os passos apropriadamente.

Parte II — Estruturas de Dados do Kernel

A segunda parte desse projeto envolve a modificação do módulo do kernel para que ele use a estrutura de dados de lista encadeada do kernel.

Na Seção 1.10, examinamos várias estruturas de dados que são comuns nos sistemas operacionais. O kernel do Linux fornece várias dessas estruturas. Aqui, examinamos o uso da lista circular duplamente encadeada que está disponível para desenvolvedores do kernel. Grande parte do que estamos discutindo está disponível no código-fonte do Linux — nessa instância, o arquivo de inclusão `<linux/list.h>` — e recomendamos que você examine esse arquivo à medida que prossegue por meio dos passos a seguir.

Inicialmente, você deve definir um `struct` contendo os elementos a serem inseridos na lista encadeada. O `struct` em C a seguir define aniversários:

```
Struct birthday {
  int day;
  int month;
  int year;
  struct list_head list;
}
```

Observe o membro `struct list_head list`. A estrutura `list_head` é definida no arquivo de inclusão `<linux/types.h>`. Sua finalidade é embutir a lista encadeada dentro dos nós que compõem a lista. A estrutura `list-head` é muito simples — ela simplesmente contém dois membros, `next` e `prev`, que apontam para a entrada posterior e anterior da lista. Ao embutir a lista encadeada dentro da estrutura, o Linux torna possível gerenciar a estrutura de dados com uma série de funções ***macro***.

Inserindo Elementos na Lista Encadeada

Podemos declarar um objeto `list_head` que usamos como referência para a cabeça da lista, empregando a macro `LIST_HEAD ( )`

```
static LIST_HEAD (birthday_list);
```

Essa macro define e inicializa a variável `birthday_list`, que é de tipo `struct list_head`.

Criamos e inicializamos instâncias de `struct birthday` como descrito a seguir:

```
struct birthday *person;

person = kmalloc(sizeof(*person), GFP KERNEL);
person->day = 2;
person->month= 8;
person->year = 1995;
```

```
INIT_LIST_HEAD(&person->list);
```

A função `kmalloc ( )` é o equivalente no kernel à função `malloc ( )`, de nível de usuário, para alocação de memória, exceto por ser a memória do kernel que está sendo alocada. (O flag `GFP_KERNE`Lindica alocação de memória de rotina no kernel.) A macro `INIT_LIST_HEAD ( )` inicializa o membro `list` em `struct birthday`. Podemos então adicionar essa instância ao fim da lista encadeada usando a macro `list_add_tail ( )`:

```
  list_add_tail(&person->list, &birthday_list);
```

Percorrendo a Lista Encadeada

Percorrer a lista encadeada envolve o uso da macro `list_for_each_entry ( )`, que aceita três parâmetros:

- Um ponteiro para a estrutura que está sendo iterada
- Um ponteiro para a cabeça da lista que está sendo iterada
- O nome da variável que contém a estrutura `list_head`

O código a seguir ilustra essa macro:

```
struct birthday *ptr;

list_for_each_entry(ptr, &birthday list, list) {
  /* a cada iteração prt aponta */
  /* para próxima estrutura birthday */
}
```

Removendo Elementos da Lista Encadeada

A remoção de elementos da lista envolve o uso da macro `list_del ( )`, que recebe um ponteiro para `struct list_head`

```
  list_del (struct list_head *element)
```

Essa macro remove *element* da lista enquanto mantém a estrutura do resto da lista.

Talvez a abordagem mais simples para a remoção de todos os elementos de uma lista encadeada seja remover cada elemento enquanto percorremos a lista. A macro `list_for_each_entry_safe ( )`comporta-se de modo semelhante a `list_for_each_entry ( )`, exceto por receber um argumento adicional que mantém o valor do ponteiro `next` do item que está sendo excluído. (Isso é necessário para preservar a estrutura da lista.) O exemplo de código a seguir ilustra essa macro:

```
struct birthday *ptr, *next

list_for_each_entry_safe(ptr,next,&birthday list,list) {
  /* a cada iteração prt aponta */
  /* para a próxima estrutura birthday */
list_del(&ptr->list);
kfree(ptr);
}
```

Observe que, após excluir cada elemento, devolvemos ao kernel, com a chamada `kfree ( )`, a memória que foi alocada anteriormente com `kmalloc ( )`. Um gerenciamento cuidadoso da memória — que inclui a liberação de memória para impedir ***vazamentos de memória*** — é crucial no desenvolvimento de código de nível de kernel.

Parte II — Tarefa

No ponto de entrada do módulo, crie uma lista encadeada contendo cinco elementos `struct birthday`. Percorra a lista encadeada e dê saída do seu conteúdo para o buffer de log do kernel. Invoque o comando `dmesg` para assegurar que a lista foi construída apropriadamente, uma vez que o módulo do kernel tenha sido carregado.

No ponto de saída do módulo, exclua os elementos da lista encadeada e devolva a memória livre ao kernel. Mais uma vez, invoque o comando dmesg para verificar se a lista foi removida, uma vez que o módulo do kernel tenha sido descarregado.