INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE SANTA CATARINA
CAMPUS SÃO JOSÉ
Componente Curricular: Sistemas Operacionais
Professor: Arliones Stevert Hoeller Junior
Aluno: João Pedro Menegali Salvan Bitencourt
Turma: 29005
Data: 27/07/2022

Módulos no Linux

Nessa atividade, foi criado um módulo para o núcleo Linux. Inicialmente, foi criado o arquivo *Makefile* abaixo:

```
obj-m = memory.o

KVERSION = $(shell uname -r)

all:
	make -C /lib/modules/$(KVERSION)/build M=$(PWD) modules
clean:
	make -C /lib/modules/$(KVERSION)/build M=$(PWD) clean
```

Em seguida, foi criado o arquivo *memory.c*, que contém o código-fonte do módulo:

```
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>       /* printk() */
#include <linux/slab.h>         /* kmalloc() */
#include <linux/fs.h>           /* everything... */
#include <linux/errno.h>        /* error codes */
#include <linux/types.h>        /* size_t */
#include <linux/proc_fs.h>
#include <linux/fcntl.h>        /* O_ACCMODE */
#include <linux/uaccess.h>      /* copy_from/to_user */

MODULE_LICENSE ("Dual BSD/GPL");

#define n 5

int memory_open (struct inode *inode, struct file *filp);
int memory_release (struct inode *inode, struct file *filp);
ssize_t memory_read (struct file *filp, char *buf, size_t count,
                     loff_t * f_pos);
ssize_t memory_write (struct file *filp, const char *buf, size_t count,
                      loff_t * f_pos);
void memory_exit (void);
int memory_init (void);

struct file_operations memory_fops =
{
  .read = memory_read,
  .write = memory_write,
  .open = memory_open,
  .release = memory_release
};

module_init (memory_init);
module_exit (memory_exit);

int memory_major = 60;
```

```
 36 int memory_major = 60;
 37 char *memory_buffer;
 38
 39 int
 40 memory_init (void)
 41 {
 42   int result;
 43   result = register_chrdev (memory_major, "memory", &memory_fops);
 44   if (result < 0)
 45     {
 46       printk ("<1>memory: cannot obtain major number %d\n", memory_major);
 47       return result;
 48     }
 49
 50   /* Allocating memory for the buffer */
 51   memory_buffer = kmalloc (1, GFP_KERNEL);
 52   if (!memory_buffer)
 53     {
 54       result = -ENOMEM;
 55       goto fail;
 56     }
 57
 58   memset (memory_buffer, 0, n);
 59   printk ("<1> Inserting memory module\n");
 60   return 0;
 61
 62 fail:
 63   memory_exit ();
 64   return result;
 65 }
 66
 67 void
 68 memory_exit (void)
 69 {
 70   unregister_chrdev (memory_major, "memory");
 71   if (memory_buffer)
 72     {
 73       kfree (memory_buffer);
 74     }
 75   printk ("<1>Removing memory module\n");
 76 }
 77
 78 int
 79 memory_open (struct inode *inode, struct file *filp)
 80 {
 81   return 0;
 82 }
 83
 84 int
 85 memory_release (struct inode *inode, struct file *filp)
 86 {
 87   return 0;
 88 }
 89
 90 ssize_t
 91 memory_read (struct file * filp, char *buf, size_t count, loff_t * f_pos)
 92 {
 93   int rv;
 94   /* Transfering data to user space */
 95   /* Changing reading position as best suits */
 96   if (*f_pos == 0)
 97     {
 98       rv=copy_to_user (buf, memory_buffer, n);
 99       if(rv)
100       {
101         printk("copy to user failed");
102         return(0);
```

```
103       }
104       *f_pos += 1;
105       return n;
106     }
107   else
108     {
109       return 0;
110     }
111 }
112
113 ssize_t
114 memory_write (struct file * filp, const char *buf, size_t count, loff_t * f_pos)
115 {
116   int rv;
117   const char *tmp;
118   tmp = buf + count - n;
119   rv=copy_from_user (memory_buffer, tmp, n);
120     if(rv)
121     {
122        printk("copy from user failed");
123        return(0);
124     }
125   *f_pos += 1;
126   return count;
127 }
```

Criado o arquivo “memory.c” com o conteúdo acima, foi utilizado o comando “make” para gerar o arquivo de módulo “memory.ko”. Em seguida, foram executados os comandos abaixo:

```
$ sudo mknod /dev/memory c 60 0
$ sudo chmod 666 /dev/memory
```

Isso criou o arquivo de dispositivo “/dev/memory” e garantiu a permissão de leitura e escrita para todos. Na sequência, o módulo recém compilado foi carregado com o comando “insmod memory.ko”. Feito isso, foi executado o comando “`echo -n "hello, MODULE" > /dev/memory`” para escrever no módulo e “`cat /dev/memory`” para leitura. Como resultado, apenas a letra “E”, que é a última da frase escrita, foi mostrada. Como a proposta era mostrar os 5 últimos caracteres, três funções precisaram ser modificadas. Inicialmente foi definido uma variável “n” valendo 5:

```
 14 #define n 5
```

Em seguida, três funções foram modicadas. Baseando-se no código A função “memory_init” era dessa forma:

```
 40 memory_init (void)
 41 {
 42   int result;
 43   result = register_chrdev (memory_major, "memory", &memory_fops);
 44   if (result < 0)
 45     {
 46       printk ("<1>memory: cannot obtain major number %d\n", memory_major);
 47       return result;
 48     }
 49
 50   /* Allocating memory for the buffer */
 51   memory_buffer = kmalloc (1, GFP_KERNEL);
 52   if (!memory_buffer)
 53     {
 54       result = -ENOMEM;
 55       goto fail;
 56     }
 57
 58   memset (memory_buffer, 0, 1);
 59   printk ("<1> Inserting memory module\n");
 60   return 0;
 61
 62 fail:
 63   memory_exit ();
 64   return result;
 65 }
```

A modificação foi feita na linha 58, onde o número 1 foi substituído por n, cujo valor é 5:

```
 40 memory_init (void)
 41 {
 42   int result;
 43   result = register_chrdev (memory_major, "memory", &memory_fops);
 44   if (result < 0)
 45     {
 46       printk ("<1>memory: cannot obtain major number %d\n", memory_major);
 47       return result;
 48     }
 49
 50   /* Allocating memory for the buffer */
 51   memory_buffer = kmalloc (1, GFP_KERNEL);
 52   if (!memory_buffer)
 53     {
 54       result = -ENOMEM;
 55       goto fail;
 56     }
 57
 58   memset (memory_buffer, 0, n);
 59   printk ("<1> Inserting memory module\n");
 60   return 0;
 61
 62 fail:
 63   memory_exit ();
 64   return result;
 65 }
```

A função “memory_read” era assim:

```
 91 memory_read (struct file * filp, char *buf, size_t count, loff_t * f_pos)
 92 {
 93   int rv;
 94   /* Transfering data to user space */
 95   /* Changing reading position as best suits */
 96   if (*f_pos == 0)
 97     {
 98       rv=copy_to_user (buf, memory_buffer, 1);
 99       if(rv)
100       {
101         printk("copy to user failed");
102         return(0);
103       }
104       *f_pos += 1;
105       return 1;
106     }
107   else
108     {
109       return 0;
110     }
111 }
```

Para obter o comportamento desejado, foram modificadas as linhas 98 e 105, onde os números uns foram substituídos por “n”, ficando dessa forma:

```
 91 memory_read (struct file * filp, char *buf, size_t count, loff_t * f_pos)
 92 {
 93   int rv;
 94   /* Transfering data to user space */
 95   /* Changing reading position as best suits */
 96   if (*f_pos == 0)
 97     {
 98       rv=copy_to_user (buf, memory_buffer, n);
 99       if(rv)
100       {
```

```
101         printk("copy to user failed");
102         return(0);
103       }
104       *f_pos += 1;
105       return n;
106     }
107   else
108     {
109       return 0;
110     }
111 }
```

Por fim, a função “memory_write” foi modificada. Esta era originalmente da forma mostrado abaixo:

```
114 memory_write (struct file * filp, const char *buf, size_t count, loff_t * f_pos)
115 {
116   int rv;
117   const char *tmp;
118   tmp = buf + count - 1;
119   rv=copy_from_user (memory_buffer, tmp, 1);
120     if(rv)
121     {
122        printk("copy from user failed");
123        return(0);
124     }
125   *f_pos += 1;
126   return count;
127 }
```

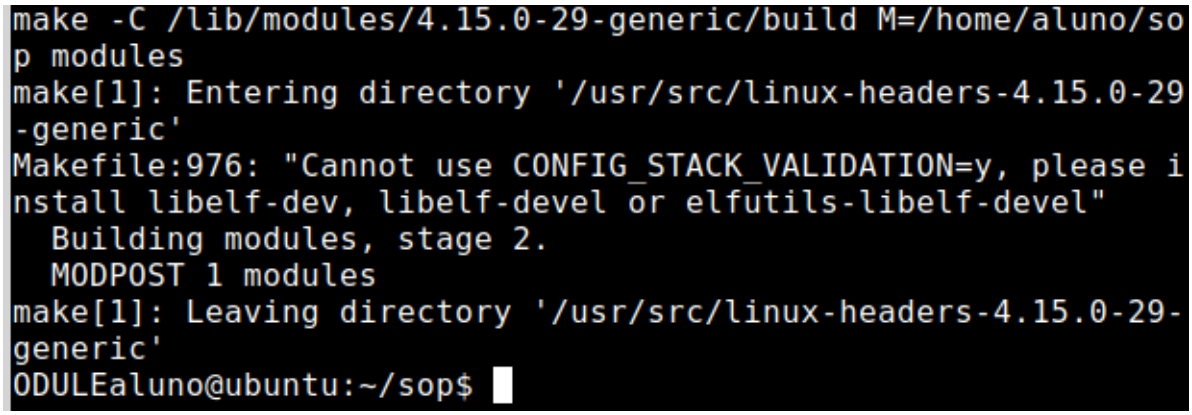
As linhas 118 e 119 foram modificadas, tendo o número 1 substituído por “n”:

```
114 memory_write (struct file * filp, const char *buf, size_t count, loff_t * f_pos)
115 {
116   int rv;
117   const char *tmp;
118   tmp = buf + count - n;
119   rv=copy_from_user (memory_buffer, tmp, n);
120     if(rv)
121     {
122        printk("copy from user failed");
123        return(0);
124     }
125   *f_pos += 1;
126   return count;
127 }
```

Finalizadas modificações, o módulo foi recompilado. O antigo módulo foi descarregado e o novo foi carregado. Ao ser escrita a frase “hello, MODULE”, o resultado lido foi “ODULE”, sendo esses os 5 caracteres finais desejados. Abaixo, está o comando que realiza a operação de descarregar o módulo antigo, compilar a versão nova, carregar o módulo ajustado, escrever a frase no dispositivo e realizar a leitura do dispositivo “/dev/memory”:

```
sudo rmmod memory.ko ; make ; sudo insmod memory.ko ; echo -n "hello, MODULE" > /dev/memory ; cat /dev/memory
```

O resultado é mostrado na imagem abaixo:



O motivo de antes da modificação apenas um caractere ter sido armazenado, é porque apenas a área de 1 byte estava sendo reservada, escrita e lida da memória. Na função “memory_init”, a área de memória é instanciada. A linha “memset (memory_buffer, 0, 1);” define o tamanho da área de memória, que era 1 e foi trocado para 5.

Na a função “memory_write” é a responsável por escrever os dados e nas linhas “tmp = buf + count – 1;” e “rv=copy_from_user (memory_buffer, tmp, 1);” eram escritos apenas 1 byte. Com os valores 1 trocados para 5, “tmp” assumiu o valor correspondente à 5 posições antes da última e a função “copy_from_user” passou a copiar os dados dessa posição indo 5 posições em diante.

Na função “memory_read”, as linha contendo “rv=copy_to_user (buf, memory_buffer, 1);” e “return 1;” foram modificadas, alterando os valores 1 por 5. Dessa forma, respectivamente, a função “copy_to_user” copia os 5 bytes do buffer e o retorno devolve esses 5 bytes para leitura.