PONTIFÍCIA UNIVERSIDADE CATÓLICA GOIÁS
CURSO DE ENGENHARIA DE COMPUTAÇÃO
ESCOLA DE CIÊNCIAS EXATAS E DA COMPUTAÇÃO

DEVICE DRIVER

GOIÂNIA 2018

PONTIFÍCIA UNIVERSIDADE CATÓLICA GOIÁS

DEVICE DRIVER

HIGOR ALVES FERREIRA
LUCAS MACEDO DA SILVA
VITOR DE ALMEIDA SILVA

GOIÂNIA, 2018

PONTIFÍCIA UNIVERSIDADE CATÓLICA GOIÁS

DEVICE DRIVER

Trabalho apresentado por Higor Alves Ferreira, Lucas Macedo da Silva, e Vitor de Almeida Silva, para avaliação e aplicação dos conceitos aprendidos em sala de aula, na disciplina de Sistemas Operacioanis I ministrada pelo Professor Doutor Cláudio Martins.

GOIÂNIA, 2018

Resumo

Device Drivers são um módulo de software que permite a comunicação do sistema operacional com o hardware em si, permitindo assim que vários dispositivos sejam incluídos ao hardware sem que seja necessário a recompilação ou reconstrução de todo um sistema operacional para a operação dos mesmos, que se trata de uma operação demorada e critica.

O desenvolvimento dos mesmos é de vital importância para qualquer profissional da área de TI.

A compilação do kernel permite a definição de parâmetros que podem vir a melhorar o desempenho do sistema de computação, já que os desenvolvedores tendem a lançar versões do kernel genéricas que sejam compatíveis com quase todos os computadores, porém podem incluir parâmetros que nunca serão utilizados.

Além de ser uma operação que permite o aprofundamento e o conhecimento sobre o kernel de sistemas Linux, bem como permite um maior controle do sistema sobre o hardware do computador.

Palavras Chaves:

- **Kernel**
- **Device Driver**
- **Major Number**

Sumário

Introdução

O desenvolvimento de um Device Driver para Linux envolve o conhecimento sobre o funcionamento do kernel do Linux, adquirido ao compilar o mesmo, que pode ser feita de forma a ser a mais apropriada para aquele dado hardware, bem como dependendo do mesmo a operação pode ser rápida ou lenta.

Um Device Driver é dos principais componentes de um sistema de computação já que permite a comunicação direta do sistema operacional com o hardware do computador, logo o conhecimento do que é necessário para o desenvolvimento de um novo Device Driver é de vital para qualquer engenheiro/cientista da computação.

O Device Driver sequencial tem como objetivo gerar números em sequência a partir do número zero, já o Device Driver Randômico tem como objetivo gerar números randômicos, a comunicação entre o usuário e os mesmos se dá a partir de um programa principal.

Desenvolvimento

Primeira Parte Compilação do Kernel

A compilação do Kernel do Linux tem como objetivo otimizar o uso do hardware de forma que a escolha de parâmetros corretos implica na otimização dos módulos instalados e uma melhora no desempenho do sistema de um modo. Porém trata-se de uma operação demorada e precisa, sendo necessário seguir corretamente os passos de compilação.

Processo adotado no desenvolvimento do trabalho: O processo de compilação do Kernel foi baseado em (SIMIONI, 2017). Sendo utilizado os hardwares:

- Intel Core i3 M 350 2,27 GHZ 4 GB de RAM, processo realizado em uma máquina virtual, com 1 GB de ram, com 1 processador, sistema operacional utilizado Lubuntu 17.10 x32 bits.
- Intel Core i5-4570T 2 4 2.9 GHz / 3.6 GHz 8 GB de RAM, processo realizado no hardware em si, sistema operacional utilizado Ubuntu 18.04 x64 bits.
- Intel Xeon 16 GB de RAM, processo realizado no hardware em si e em uma máquina virtual com 8 GB de RAM, sistema operacional utilizado Ubuntu 18.04 x64 bits.

A compilação se fez de forma mais rápida no hardware em si, já que não existem virtualizações logo o processo se torna mais rápido, bem como não existem restrições de uso dos recursos disponíveis.

Segunda Parte Desenvolvimento do Device Driver

Device Drivers são módulos de software que permitem a comunicação entre o sistema operacional e o hardware, sendo portanto de extrema importância para o perfeito funcionamento do sistema de computação.

Foram desenvolvidos dois drivers que se comunicam com o Kernel são eles:

- **SequencialN:** Gera um número sequencial a cada vez que uma operação de escrita é efetuada, começando a partir de zero inicializado na função init.
- **RND:** Gera um número aleatório a cada operação de escrita, sendo gerado em conjunto com a biblioteca random.h, começando também com um número aleatório, gerado pela biblioteca random.h.

SequencialN

Descrição Geral: Gera um número em sequência a cada operação de escrita, opção 1, no programa principal, começando a partir do número 0, até n, conforme o usuário quiser. A visualização do número sequencial gerado se dá a partir da operação de leitura disponível pelo Device Driver, opção 2 no programa principal.

Descrição das Funções:

- **static ssize_t seqN_read (struct file *, char *, size_t , loff_t *):** Realiza a operação de leitura do último número sequencial gerado pelo Device Driver. Recebendo como parâmetros:
    - 1)Tipo: file, Ponteiro para o objeto arquivo, definido pelo sistema de arquivos do Linux.
    - 2) Tipo char, Ponteiro para o buffer, variável que conterá o conteúdo a ser lido.
    - 3) Tipo size_t, Contém o tamanho do buffer.
    - 4) Tipo lofft_t, Contém o deslocamento no arquivo.

- **static int seqN_open (struct inode *, struct file *):** Responsável por abrir o arquivo do Device Driver sempre que ele for ser utilizado. Recebendo como parâmetros:
    - 1) Tipo: inode: Ponteiro para o objeto inode, definido no sistema de arquivos do Linux.
    - 2) Tipo file: Ponteiro para o objeto arquivo, definido no sistema de arquivos do Linux.

- **static int seqN_release (struct inode *, struct file*):** Responsável por fechar ou liberar, ou seja, é chamada sempre que o programa que está utilizando o arquivo do Driver é fechado. Recebendo como parâmetros:
    - 1) Tipo inode: Ponteiro para o objeto inode, definido no sistema de arquivos do Linux.
    - 2) Tipo file: Ponteiro para o objeto arquivo, definido no sistema de arquivos do Linux.

- **static ssize_t seqN_write(struct file *, const char *, size_t , loff_t *):** Responsável por ler o último número sequencial gerado. Recebendo como parâmetros:
    - 1)Tipo: file, Ponteiro para o objeto arquivo, definido pelo sistema de arquivos do Linux.
    - 2) Tipo char, Ponteiro para o buffer, variável que conterá o conteúdo a ser lido.
    - 3) Tipo size_t, Contém o tamanho do buffer.
    - 4) Tipo lofft_t, Contém o deslocamento no arquivo.

- **static int seqN_init(void):** Executada sempre que o Device Driver é carregado para a memória, sendo responsável por permitir a perfeita operação do mesmo, inicializando a variável k com zero, registrando o Device Driver, alocando o major number para o mesmo. Não recebi nenhum parâmetro.

- **static void seqN_exit(void):** Executada sempre que o Device Driver é removido da memória, sendo responsável por retirar o registro do Device Driver, bem como desalocar o major number do mesmo.

RND

Descrição Geral: Gera um número aleatório a cada operação de escrita, opção 1, no programa principal, começando a partir do número de um número aleátorio, até n, conforme o usuário quiser. A visualização do número aleatório gerado se dá a partir da operação de leitura disponível pelo Device Driver, opção 2 no programa principal.

Descrição das Funções:

- **static ssize_t seqN_read (struct file *, char *, size_t , loff_t *):** Realiza a operação de leitura do último número aleatório gerado pelo Device Driver. Recebendo como parâmetros:
    - 1)Tipo: file, Ponteiro para o objeto arquivo, definido pelo sistema de arquivos do Linux.
    - 2) Tipo char, Ponteiro para o buffer, variável que conterá o conteúdo a ser lido.
    - 3) Tipo size_t, Contém o tamanho do buffer.
    - 4) Tipo lofft_t, Contém o deslocamento no arquivo.

- **static int seqN_open (struct inode *, struct file *):** Responsável por abrir o arquivo do Device Driver sempre que ele for ser utilizado. Recebendo como parâmetros:
    - 1) Tipo: inode: Ponteiro para o objeto inode, definido no sistema de arquivos do Linux.
    - 2) Tipo file: Ponteiro para o objeto arquivo, definido no sistema de arquivos do Linux.

- **static int seqN_release (struct inode *, struct file*):** Responsável por fechar ou liberar, ou seja, é chamada sempre que o programa que está utilizando o arquivo do Driver é fechado. Recebendo como parâmetros:
    - 1) Tipo inode: Ponteiro para o objeto inode, definido no sistema de arquivos do Linux.
    - 2) Tipo file: Ponteiro para o objeto arquivo, definido no sistema de arquivos do Linux.
    -
- **static ssize_t seqN_write(struct file *, const char *, size_t , loff_t *):** Responsável por ler o último número sequencial gerado. Recebendo como parâmetros:
    - 1)Tipo: file, Ponteiro para o objeto arquivo, definido pelo sistema de arquivos do Linux.

- 2) Tipo char, Ponteiro para o buffer, variável que conterá o conteúdo a ser lido.
- 3) Tipo size_t, Contém o tamanho do buffer.
- 4) Tipo lofft_t, Contém o deslocamento no arquivo.

- **static int seqN_init(void):** Executada sempre que o Device Driver é carregado para a memória, sendo responsável por permitir a perfeita operação do mesmo, inicializando a variável k com um número aleatorio, registrando o Device Driver, alocando o major number para o mesmo. Não recebi nenhum parâmetro.

- **static void seqN_exit(void):** Executada sempre que o Device Driver é removido da memória, sendo responsável por retirar o registro do Device Driver, bem como desalocar o major number do mesmo.

Crypto

Descrição Geral: Gera um arquivo para ser criptografado, na opção 1 do programa principal, para criptografa-lo utiliza-se a opção 3, para ler o arquivo gerado utiliza-se a opção 2 do programa, para descriptografar utiliza-se a opção 4 do programa principal. Vale ressaltar que a opção de leitura tem dois possíveis resultados a leitura do arquivo criptografado e o arquivo descriptografado, bem como as operações definidas em 3 e 4 requerem a senha do root, além de o arquivo de saída .bin.

Portanto o programa tem como entrada um arquivo .txt e gera um arquivo de saída .bin, com os arquivos criptografados.

Todas as operações realizadas são possíveis a partir da API disponível no Linux OpenSSL.

Terceira parte compilação e carga do Device Driver

A compilação do Device Driver e do programa principal que permite a comunicação do usuário com o mesmo foi realizada, a partir do arquivo makefile, que permite a realização desta tarefa mais facilmente, definindo as regras de compilação da mesma, bem como realizar as ligações necessárias em algumas bibliotecas, o mesmo se encontra no Apêndice A – Makefile. Para executá-lo é possível a partir do comando “make”.

A carga do Device Driver é realizada pelo comando “insmod ./NomeDoDeviceDriver.ko”, após isso, torna-se possível o uso da programa principal que permite a comunicação entre o usuário e o Device Driver, sendo compilada com o comando “./NomePrograma principal”. A remoção do Device Driver é realizada pelo comando “rmmod NomeDoDeviceDriver”. A visualização das mensagens geradas durante a execução do Device Driver pode ser vista a partir do comando “dmesg”.

Portanto pode se definir a sequência de passos para a compilação do device Driver conforme abaixo, onde DD indica o nome do Device Driver, main o nome do programa principal.

```
# make
# insmod ./DD.ko
#./main
# rmmod DD
# dmesg
```

O comando “ls -l /dev”, permite a visualização dos Device Drivers do sistema, bem como do SequencialN ou RND, se tiverem sido carregados.

No caminho “/proc/devices”, é possível verificar os Device Drivers do sistemas, além de seus respectivos major number, o arquivo devices é composto pela especificação do Device Driver, character ou block device, bem como cada linha contém o major number do Device e seu nome.

Conclusão

Portanto dado a extrema importância de um Device Driver em um sistema de computação bem como em conjunto com a compilação do kernel do Linux definida para aquele hardware, o desenvolvimento de novos módulos por parte de qualquer desenvolvedor se torna mais simples.

A operação de compilação do kernel é lenta e critica, se dando de forma mais certa e menos uniforme quando executada no hardware em si, além de gastar menos tempo do que em relação a operação realizada em máquinas virtuais, que limitam o uso do harware, bem como o hardware fornecido é mais uniforme, fazendo com que a devida exploração de um kernel se dê de forma mais expandida a partir de uma máquina física.

O desenvolvimento de um Device Driver requer o conhecimento prévio do sistema Linux, seu kernel, sistema de arquivos e comandos que são de extrema importância durante o desenvolvimento. Podendo ser do tipo character ou em blocos diferindo apenas na forma em que os dados são transmitidos, caracter a caracter, ou por blocos respectivamente.

Portanto, é de extrema importância tais conhecimentos para futuras aplicações ou desenvolvimento de novas tecnologias, já que a maioria dos embarcados, trabalha diretamente com o kernel, tornando se assim o uso de módulos do Kernel para permitir a comunicação entre o hardware e a interface de aplicação.

Referências Bibliográficas

ALMEIDA, Rubens Queiroz de. **Examinando as mensagens do Kernel GNU/Linux.** 2012. Disponível em: <http://www.dicas-l.com.br/arquivo/examinando_as_mensagens_do_kernel_gnu_linux.php#.WzEAZtJKjIV>. Acesso em: 21 jun. 2018.

ARAÚJO, Pedro. **CRIANDO LINKS PARA ARQUIVOS E DIRETÓRIOS NO LINUX.** 2010. Disponível em: <https://www.vivaolinux.com.br/dica/Criando-links-para-arquivos-e-diretorios-no-Linux>. Acesso em: 21 jun. 2018.

CALBET, Xavier. **Breve tutorial para escribir drivers en Linux.** 2001. Disponível em: <http://www.exa.unicen.edu.ar/catedras/rtlinux/material/apuntes/driv_tut_last.pdf>. Acesso em: 22 jun. 2018.

CANIN, William. **Linux: Compilando e instalando o kernel em modo tradicional.** 2015. Disponível em: <https://williamcanin.me/blog/linux-compilando-e-instalando-o-kernel-em-modo-tradicional/>. Acesso em: 19 jun. 2018.

CENTER, Ibm Knowledge. **Device names, device nodes, and major/minor numbers.** Disponível em: <https://www.ibm.com/support/knowledgecenter/en/linuxonibm/com.ibm.linux.z.lgdd/lgdd_c_udev.html>. Acesso em: 21 jun. 2018.

CORBERT, Jonathan; RUBINI, Alessandro; KROAH-HARTMAN, Greg. **Linux Device Driver.** 3. ed. Cambridge: O Reilly, 2005.

DAY, Rob. **The Kernel Newbie Corner: Kernel Debugging with proc "Sequence" Files--Part 2.** 2009. Disponível em: <https://www.linux.com/learn/kernel-newbie-corner-kernel-debugging-proc-sequence-files-part-2>. Acesso em: 21 jun. 2018.

KROAH-HARTMAN, Greg. **How to Write a Linux USB Device Driver.** 2001. Disponível em: <https://www.linuxjournal.com/article/4786>. Acesso em: 22 jun. 2018.

MAGALHÃES, Cristiano Meira. **Saiba o que são e como instalar drivers em um sistema com Linux.** 2008. Disponível em: <http://pcworld.com.br/dicas/2008/12/05/saiba-o-que-sao-e-como-instalar-drivers-em-um-sistema-com-linux/>. Acesso em: 21 jun. 2018.

MOLLOY, Derek. **Writing a Linux Kernel Module — Part 2: A Character Device.** Disponível em: <http://derekmolloy.ie/writing-a-linux-kernel-module-part-2-a-character-device/>. Acesso em: 20 jun. 2018.

MORIMOTO, Carlos. **Entendendo o Linux:** O Kernel. 2010. Disponível em: <https://www.hardware.com.br/guias/entendendo-linux/kernel.html>. Acesso em: 21 jun. 2018.

PRADO, Sergio. **Linux Device Drivers – Parte 1.** Disponível em: <https://sergioprado.org/linux-device-drivers-parte-1/>. Acesso em: 20 jun. 2018.

PRADO, Sergio. **Linux Device Drivers – Parte 2.** Disponível em: <https://sergioprado.org/linux-device-drivers-parte-1/>. Acesso em: 20 jun. 2018.

PRADO, Sergio. Mini2440 – Compilando aplicações e device drivers**.** Disponível em: <https://sergioprado.org/linux-device-drivers-parte-1/>. Acesso em: 20 jun. 2018.

SALZMAN, Peter Jay; BURIAN, Michael; POMERANTZ, Ori. **The Linux Kernel Module Programming Guide.** 2007. Disponível em: <http://tldp.org/LDP/lkmpg/2.6/html/index.html>. Acesso em: 21 jun. 2018.

SIMIONI, Dionatan. **Como compilar um Kernel Linux passo a passo [TUTORIAL COMPLETO].** Disponível em: <https://www.diolinux.com.br/2017/07/como-compilar-um-kernel-linux-passo-a-passo.html>. Acesso em: 19 jun. 2018.

VAHALA, Uresh. **UNIX Internals the new frontier.** New Jersey: Prentice Hall, 1996.

Apêndice A

Makefile

```
obj-m+=DD.o

all:
	make -C /lib/modules/$(shell uname -r)/build/ M=$(PWD) modules
	$(CC) main.c -o main
clean:
	make -C /lib/modules/$(shell uname -r)/build/ M=$(PWD) clean
	rm main
```

Apêndice B

SequencialN

```
#include <linux/init.h>
#include <linux/module.h>
#include <linux/device.h>
#include <linux/kernel.h>
#include <linux/fs.h>
#include <linux/uaccess.h>
#include <openssl/sha.h>

// ---------------------- Váriaveis Globais -----------------------------
// São declaradas estaticas para serem globais dentro do arquivo.

static int majorNumber;
static int k;
static struct class * seqClass = NULL;
static struct device * seqDev = NULL;
static char mensagem[256] = {0};
static short tam_mensagem;
static int qtdAberturas = 0;


#define NAME "SequencialN"
#define CLASS_NAME "Seq"
MODULE_LICENSE("Dual BSD/GPL");


// ---------------------- Cabeçalho das funções -------------------------

static ssize_t seqN_read (struct file *, char *, size_t , loff_t *);
static int seqN_open (struct inode *, struct file *);
static int seqN_release (struct inode *, struct file*);
static ssize_t seqN_write(struct file *, const char *, size_t , loff_t *);

static void hello_exit(void);
static int hello_init(void);



// ------------------- Estrutura com as operações de arquivos ----------
```

```
static struct file_operations fops = {  //As operações não declaradas são consideradas como NULL
        .owner = THIS_MODULE,
        .read = seqN_read,
        .open = seqN_open,
        .release = seqN_release,
        .write = seqN_write,
};




// ------------------ Funções e Métodos -------------------------------

// --- Operações com arquivo
static ssize_t seqN_read (struct file * filp, char * buffer, size_t lenght, loff_t * offset){
        k ++;

        int var = copy_to_user(buffer, mensagem, tam_mensagem);

        printk ("SequencialN: Lendo o número (%s) gerado pelo Device Driver \n", mensagem);
        printk ("SequencialN: K vale -> (%i)\n", k);

        tam_mensagem = 0;
        if(var == 0)
        {
                printk ("SequencialN: Sucesso ao realizar a operação de leitura\n");
                return k;
        }else
        {
                printk ("SequencialN: Erro ao realizar a operação de leitura\n");
                return -EFAULT; //Caso receba um local de memória inválido
        }

}

static int seqN_open (struct inode * inode, struct file * file){
        qtdAberturas ++;
        printk ("SequencialN: Arquivo aberto\n");
        printk ("SequencialN: O arquivo foi aberto (%i) vezes\n",  qtdAberturas);
        return 0;
}
```

```
static int seqN_release(struct inode * inode, struct file * filep )
{
	printk ("SequencialN: Device Driver, fechado com sucesso\n");
	return 0;
}

static ssize_t seqN_write(struct file *filep, const char *buffer, size_t len, loff_t *offset){
  char buf = (char)k;
  sprintf(mensagem, "%d" ,buf);   //Armazenando o que foi recebido em mensagem
  tam_mensagem = strlen(mensagem);
  printk ("SequencialN: Escrevendo o número -> (%s) gerado pelo Device Driver", mensagem);
  printk ("SequencialN: Operação de escrita realizada com sucesso\n");
  return len;
}

// --- init e exit
static int hello_init(void)
{
	printk (KERN_ALERT "SequencialN: Inicializando a operação do Device Driver\n");

	k = 0;

	//Alocando o major number do device driver
	majorNumber = register_chrdev (0, NAME, &fops);
	if (majorNumber < 0){
		printk ("SequencialN: Erro ao alocar o major number para o Device Driver\n");
		return majorNumber;
	}
	printk ("SequencialN: Alocado o major number -> (%i) para o Device Driver\n", majorNumber);


	//Registrando a classe do Device Driver
	seqClass = class_create (THIS_MODULE, CLASS_NAME);
	if (IS_ERR(seqClass)){
		unregister_chrdev (majorNumber, NAME);
		printk ("SequencialN: Desalocado o major number do Device Driver\n");
		printk ("SequencialN: Não foi possivel registrar a classe do Device Driver\n");

		return -1;
	}
```

```
	//Registrando o Device Driver
	seqDev = device_create(seqClass, NULL, MKDEV(majorNumber, 0), NULL, NAME);
	if(IS_ERR(seqDev)){
		class_destroy(seqClass);
		unregister_chrdev (majorNumber, NAME);
		printk ("SequencialN: Desalocado o major number do Device Driver\n");
		printk ("SequencialN: Não foi possivel registrar o Device Driver\n");
		return -1;
	}

	printk ("SequencialN: Device Driver criado com sucesso\n");

	return 0;
}

static void hello_exit(void)
{
	//Desalocando o major number do Device Driver
	unregister_chrdev (majorNumber, NAME);
	printk ("SequencialN: Desalocado o major number do Device Driver\n");

	device_destroy(seqClass, MKDEV(majorNumber, 0));     // remove the device
	class_unregister(seqClass);                          // unregister the device class
	class_destroy(seqClass);

	printk(KERN_ALERT "SequencialN: Finalizada a execução do Device Driver, Adeus mundo cruel\n\n");
}


module_init(hello_init);
module_exit(hello_exit);


// -------- END -------
```

Apêndice C

Programa Principal – SequencialN

```
/*
 * Autores: Higor Alves Ferreira, Lucas Macedo da Silva, Vitor de Almeida Silva.
 * Versão: 2.3.1
 *
 * SequencialN - Gera números Sequênciais.
 *
 */


#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <fcntl.h>
#include <unistd.h>
#include <string.h>


#define TAM_FROM 256

int main (){
	int arq, var;
	int c = 0;

	char from[TAM_FROM] = {0}, msg[TAM_FROM];

	arq = open ("/dev/SequencialN", O_RDWR);
	if (arq < 0){
		printf("Erro, arquivo não aberto\n");
		return 0;
	}

	printf ("Arquivo aberto com sucesso\n");
	system("clear");

	do{
		var = 0;
```

```
		printf("\nDigite uma opção: \n[1] Gerar número sequencial \n[2] Ler número gerado \n[3] Help \n[0] Sair\n");
		scanf ("%i", &c);

		switch (c){
			case 0:
				system("clear");
				return 0;
				break;

			case 1:

				var = write (arq, NULL, 0);
				if (var < 0){
					printf ("Erro, operação de escrita não realizada\n");
				}else{
					printf("Operação de escrita efetuada com sucesso\n");
				}
				break;

			case 2:
				var = read (arq, from, TAM_FROM);
				if (var < 0){
					printf ("Erro, operação de leitura não realizada\n");
				}else{
					printf ("Foi lido: %s \n", from);
				}
				break;

			case 3:
				printf ("\n\n ---Device Driver Help --- \n Autores: Higor Alves, Lucas Macedo, Vitor de Almeida.\n\nOpção [1] comunica com o Device Driver para que o mesmo gere um número aleatório com a ajuda da biblioteca random.h \nOpção [2] comunica com o Device Driver que seja lido o último número aleatório gerado\n");
				break;


			default:
				printf ("Erro, informe uma opção válida\n");
				printf("\nDigite uma opção: \n[1] Escrever \n[2] Ler \n[3] Help\n[0] Sair\n");
				scanf ("%i", &c);
				break;
		}
```

```
        }while(c != 0);
        system("clear");

        return 0;

}
```

Apêndice D

RND

```
#include <linux/init.h>
#include <linux/module.h>
#include <linux/device.h>
#include <linux/kernel.h>
#include <linux/fs.h>
#include <linux/uaccess.h>
#include <linux/random.h>    //Usada para gerar os números aleatorios

// ---------------------- Váriaveis Globais -----------------------------
// São declaradas estaticas para serem globais dentro do arquivo.

static int majorNumber;
static int k;
static struct class * seqClass = NULL;
static struct device * seqDev = NULL;
static char mensagem[256] = {0};
static short tam_mensagem;
static int qtdAberturas = 0;


#define NAME "RND"
#define CLASS_NAME "RND"
MODULE_LICENSE("Dual BSD/GPL");


// ---------------------- Cabeçalho das funções -------------------------

static ssize_t RND_read (struct file *, char *, size_t , loff_t *);
static int RND_open (struct inode *, struct file *);
static int RND_release (struct inode *, struct file*);
static ssize_t RND_write(struct file *, const char *, size_t , loff_t *);

static void RND_exit(void);
static int RND_init(void);


// ------------------- Estrutura com as operações de arquivos ----------
```

```
static struct file_operations fops = {  //As operações não declaradas são consideradas como NULL
        .owner = THIS_MODULE,
        .read = RND_read,
        .open = RND_open,
        .release = RND_release,
        .write = RND_write,
};

// ------------------ Funções e Métodos -------------------------------

// --- Operações com arquivo
static ssize_t RND_read (struct file * filp, char * buffer, size_t lenght, loff_t * offset){
        //k ++;

        int var = copy_to_user(buffer, mensagem, tam_mensagem);

        tam_mensagem = 0;
        if(var == 0)
        {
                printk ("RND: Sucesso ao realizar a operação de leitura\n");
                return 0;
        }else
        {
                printk ("RND: Erro ao realizar a operação de leitura\n");
                return -EFAULT; //Caso receba um local de memória inválido
        }

}

static int RND_open (struct inode * inode, struct file * file){
        qtdAberturas ++;
        printk ("RND: Arquivo aberto\n");
        printk ("RND: O arquivo foi aberto (%i) vezes\n", qtdAberturas);
        return 0;
}

static int RND_release(struct inode * inode, struct file * filep )
{
        printk ("RND: Device Driver, fechado com sucesso\n");
        return 0;
}
```

```
static ssize_t RND_write(struct file *filep, const char *buffer, size_t len, loff_t *offset){
   k = (get_random_int() + get_random_int());
   char buf = (char) k;
   sprintf(mensagem, "%d" ,buf);   //Armazenando o que foi recebido em mensagem
   tam_mensagem = strlen(mensagem);
   printk ("RND: Foi gerado o número (%s) pelo device Driver", mensagem);
   printk("RND: Operação de escrita realizada com sucesso\n");
   return len;
}

// --- init e exit
static int RND_init(void)
{
        printk (KERN_ALERT "RND: Inicializando a operação do Device Driver\n");

        k = (get_random_int() + get_random_int());

        printk ("RND: K vale -> (%i)\n", k);


        //Alocando o major number do device driver
        majorNumber = register_chrdev (0, NAME, &fops);
        if (majorNumber < 0){
                printk ("RND: Erro ao alocar o major number para o Device Driver\n");
                return majorNumber;
        }
        printk ("RND: Alocado o major number -> (%i) para o Device Driver\n",
majorNumber);


        //Registrando a classe do Device Driver
        seqClass = class_create (THIS_MODULE, CLASS_NAME);
        if (IS_ERR(seqClass)){
                unregister_chrdev (majorNumber, NAME);
                printk ("RND: Desalocado o major number do Device Driver\n");
                printk ("RND: Não foi possivel registrar a classe do Device Driver\n");

                return -1;
        }


        //Registrando o Device Driver
        seqDev = device_create(seqClass, NULL, MKDEV(majorNumber, 0), NULL,
NAME);
```

```
	if(IS_ERR(seqDev)){
		class_destroy(seqClass);
		unregister_chrdev (majorNumber, NAME);
		printk ("RND: Desalocado o major number do Device Driver\n");
		printk ("RND: Não foi possivel registrar o Device Driver\n");
		return -1;
	}

	printk ("RND: Device Driver criado com sucesso\n");

	return 0;
}

static void RND_exit(void)
{
	//Desalocando o major number do Device Driver
	unregister_chrdev (majorNumber, NAME);
	printk ("RND: Desalocado o major number do Device Driver\n");

	device_destroy(seqClass, MKDEV(majorNumber, 0));     // remove the device
	class_unregister(seqClass);                          // unregister the device class
	class_destroy(seqClass);

	printk(KERN_ALERT "RND: Finalizada a execução do Device Driver\n\n");
}

module_init(RND_init);
module_exit(RND_exit);

// -------- END -------
```

Apêndice E

Programa Principal – RND

```
/*
 * Autores: Higor Alves Ferreira, Lucas Macedo da Silva, Vitor de Almeida Silva.
 * Versão: 1.1
 *
 * RND - Gera números Aleátorios.
 *
 */


#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <fcntl.h>
#include <unistd.h>
#include <string.h>
#include <time.h>
#include <string.h>


#define TAM_FROM 256

int main (){
	int arq, var;
	int c = 0;

	char from[TAM_FROM] = {0}, msg[TAM_FROM];

	arq = open ("/dev/RND", O_RDWR);
	if (arq < 0){
		printf("Erro, arquivo não aberto\n");
		return 0;
	}

	printf ("Arquivo aberto com sucesso\n");
	system("clear");

	do{
```

```
		var = 0;
		printf("\nDigite uma opção: \n[1] Gerar número aleátorio \n[2] Ver número gerado \n[3] Help \n[0] Sair\n");
		scanf ("%i", &c);

		switch (c){
			case 0:
				system("clear");
				return 0;
				break;

			case 1:

				var = write (arq, "", 0);
				printf("Número aleatorio gerado com sucesso\n");
				break;

			case 2:
				var = read (arq, from, TAM_FROM);
				printf ("Foi gerado o n: %s \n", from);

				break;

			case 3:
				printf ("\n\n ---Device Driver Help --- \n Autores: Higor Alves, Lucas Macedo, Vitor de Almeida.\n\nOpção [1] comunica com o Device Driver para que o mesmo gere um número aleatório com a ajuda da biblioteca random.h \nOpção [2] comunica com o Device Driver que seja lido o último número aleatório gerado\n");
				break;


			default:
				printf ("Erro, informe uma opção válida\n");
				printf("\nDigite uma opção: \n[1] Escrever \n[2] Ler \n[3] help \n[0] Sair\n");
				scanf ("%i", &c);
				break;
		}
	}while(c != 0);
	system("clear");

	return 0;
```

Apêndice F

Crypto

```
/*
 * Autores: Higor Alves Ferreira, Lucas Macedo da Silva, Vitor de Almeida Silva.
 * Versão: 1.2
 *
 * Crypto - Gera um arquivo criptografado.
 *
 */

#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <fcntl.h>
#include <unistd.h>
#include <string.h>
//#include <openssl/evp.h>


#define TAM_FROM 256

int main (){
	int arq, var;
	int c = 0;

	char from[TAM_FROM] = {0},semCript[TAM_FROM] = {0}, msg[TAM_FROM], comCript[TAM_FROM]= {0};
	char cmd[255] = {0};


	FILE *ptr_arquivo;


	arq = open ("/dev/SequencialN", O_RDWR);
	if (arq < 0){
		printf("Erro, arquivo não aberto\n");
		return 0;
	}
```

```
	printf ("Arquivo aberto com sucesso\n");
	system("clear");

	do{
		var = 0;
		printf("\nDigite uma opção: \n[1] Escrever \n[2] Ler \n[3] Criptografar \n[4] Descriptografar \n[5] Help \n[0] Sair\n");
		scanf ("%i", &c);

		switch (c){
			case 0:
				system("clear");
				return 0;
				break;

			case 1:

				var = write (arq, NULL, 0);
				if (var < 0){
					printf ("Erro, operação de escrita não realizada\n");
				}else{
					printf("Operação de escrita efetuada com sucesso\n");
				}

				printf("\nInsira uma String, Para ser criptografada:");
				scanf("%s",semCript);

				sprintf(cmd,"%s" "%s" "%s", "echo ", semCript, " >string.txt");
				system(cmd);
				//system("openssl enc -e -des3 -salt -in string.txt -out cripto.bin");
				//system("openssl enc -d -des3 -in cripto.bin -out string2.txt");



				break;

			case 2:

				/*var = read (arq, from , TAM_FROM);
				if (var < 0){
					printf ("Erro, operação de leitura não realizada\n");
				}else{
```

```
                printf ("Foi lido: %s \n", cmd);
        }

        */

        ptr_arquivo=NULL;
        ptr_arquivo = fopen("string.txt", "r");
        system("clear");

        if(ptr_arquivo )
        {
                //system("hd string.txt");
                system("cat string.txt");
        }
        else
        {
                system("cat string.bin");
        }

        break;


case 3:
        system("openssl enc -e -des3 -salt -in string.txt -out string.bin"); //Realiza a criptografia do arquivo string.txt
        system("rm -r string.txt");
                        //Gerando o arquivo string.bin
        break;

case 4:
        system("openssl enc -d -des3 -in string.bin -out string.txt");
        break;

case 5:
        printf ("\n\n ---Device Driver Help --- \n Autores: Higor Alves, Lucas Macedo, Vitor de Almeida.\n\nOpção [1] comunica com o Device Driver para que o mesmo gere um número aleatório com a ajuda da biblioteca random.h \nOpção [2] comunica com o Device Driver que seja lido o último número aleatório gerado\n");
        break;

default:
        printf ("Erro, informe uma opção válida\n");
        printf("\nDigite uma opção: \n[1] Escrever \n[2] Ler \n[3] Criptografar \n[4] Descriptografar \n[5] Help \n[0] Sair\n");
```

```
                        scanf ("%i", &c);
                        break;
                }
        }while(c != 0);
        system("clear");

        return 0;

}
```