



Pontifícia Universidade Católica de Campinas
Faculdade de Engenharia de Computação -
FECOMP

Sistemas Operacionais B – Relatório Atividade 2
Compilação de módulos do kernel Linux

Beatriz Morelatto Lorente RA: 18071597

Cesar Marrote Manzano RA: 18051755

Fabricio Silva Cardoso RA: 18023481

Pedro Ignácio Trevisan RA: 18016568

Sumário

1.Introdução.....	3
2.Conceitos iniciais.....	4
3.Principais passos da compilação de um módulo.....	5
4.Conclusão.....	13

Introdução

O experimento desenvolvido pretende demonstrar os detalhes da implementação de um módulo do kernel Linux além de se familiarizar com a programação dentro do kernel. Todo o processo de instalação e compilação de um módulo será explicado ao longo do relatório e é necessário compreender cada passo para se obter um resultado satisfatório.

Conceitos iniciais

Antes de demonstrar o processo de implementação de um módulo do kernel Linux, é importante esclarecer alguns pontos para que o entendimento do experimento fique mais fácil.

Alguns detalhes sobre os módulos

Os módulos que vamos trabalhar são aqueles que são carregados no kernel em tempo de execução, ou seja, estendem a funcionalidade do kernel em tempo de execução.

Normalmente os programas feitos em C, começam com a função `main()`, executando vários passos e encerrando sua execução. Já um módulo, quando é carregado para dentro do kernel, ele não é executado imediatamente, ou seja, ele apenas avisa o kernel que está disponível para ser chamado quando precisar. Isto é feito através de uma função de entrada, que mostra sua funcionalidade para o kernel. Os módulos também possuem uma função para desfazer tudo que foi feito pela função de entrada, uma função de saída, liberando recursos e afins, quando este for retirado do kernel.

Os módulos que serão apresentados serão básicos e suas funcionalidades não são muito práticas, serão mostrados apenas para compreensão do processo de compilação e de alguns conceitos de programação em espaço de kernel.

Espaço de usuário e espaço de kernel

Quando nos referimos ao espaço de kernel, falamos de um lugar onde não há restrições, ou seja, uma aplicação rodando nesse espaço tem acesso total à máquina desde a memória até o controle do hardware do computador. Todos os módulos que serão mostrados rodam nesse espaço. Já o espaço de usuário é mais restritivo, menos privilegiado e há recursos que são limitados para as aplicações que rodam nesse espaço. Esses níveis servem para garantir segurança ao sistema operacional, impedindo que programas de terceiros venham a prejudicar o funcionamento normal da máquina.

Programação em espaço de kernel

Ao se programar em espaço de kernel, também há diversas restrições, portanto, ao programar um módulo haverá algumas diferenças no modo de projetá-lo.

Uma diferença notável são as bibliotecas usadas. Uma vez que o kernel roda por si só, não há como usar as bibliotecas padrão da linguagem C, como a `'stdlib'`, `'stdio'`, por exemplo. Todas as funções usadas em um módulo são aquelas que vem com o kernel, ou seja, as únicas bibliotecas que podem ser usadas são fornecidas pelo mesmo.

Principais passos da compilação de um módulo

Para iniciarmos o processo de instalação e compilação de um módulo do kernel Linux, é importante preparar o ambiente e já ter conhecimento de alguns aspectos do kernel. Para o experimento foi usado a versão do kernel 4.15.0 e o Ubuntu 16.04. Todo o processo foi feito através do terminal do Linux.

É recomendado que todo o processo seja feito no modo 'root' da máquina. Não é algo obrigatório, apenas garante que todo o processo será feito sem eventuais erros. Para isso basta digitar o comando `'sudo su'` em seu terminal.

Inicialmente é necessário instalar a biblioteca 'build-essential', que conterà alguns comandos essenciais para a compilação de um módulo. Basta digitar o comando `'sudo apt-get install build-essential kmod'` ou apenas `'sudo apt-get install build-essential'` em seu terminal.

Após instalar a biblioteca é necessário instalar os 'headers files' para o seu sistema. Isso possibilitará que o módulo tenha acesso às funções oferecidas pelo kernel. Antes de prosseguir, é importante atualizar algumas coisas no seu kernel com o comando `'sudo apt-get update'`. Antes de instalar é necessário pesquisar algumas das versões disponíveis, com o comando `'apt-cache search linux-headers-$(uname -r)'`. Caso o comando não retorne nada, significa que você já tem uma versão instalada. Caso contrário, basta instalar com o comando `'sudo apt-get install linux-headers-xxx'` (sendo 'xxx' a versão que você deseja).

A partir desse ponto será mostrado a compilação dos módulos, de fato. Para acessar o código fonte dos módulos apresentados, basta acessar o link abaixo:

<https://github.com/cesarmmanzano/Atividades-SO-B/tree/master/Atividade%202/modulos>

Primeiro módulo: hello-1

Para qualquer módulo é necessário fazer um arquivo com o nome 'Makefile'. Isso é necessário para que o módulo possa ser compilado. Para isso, basta abrir um editor de sua preferência e copiar o texto da imagem abaixo:

```
obj-m += hello-1.o

all:
    make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules

clean:
    make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

Figura 1 - Arquivo Makefile do primeiro módulo

Após fazer o arquivo, basta digitar o comando `'make'` em seu terminal, na pasta na qual o seu módulo está inserido, para a compilação iniciar.

```
root@cesar-VirtualBox: /home/cesar/Documentos/programas/hello1
root@cesar-VirtualBox:/home/cesar/Documentos/programas/hello1# make
make -C /lib/modules/4.15.18/build M=/home/cesar/Documentos/programas/hello1 mod
ules
make[1]: Entering directory '/home/cesar/Downloads/linux-source-4.15.0'
Building modules, stage 2.
MODPOST 1 modules
WARNING: modpost: missing MODULE_LICENSE() in /home/cesar/Documentos/programas/h
ello1/hello-1.o
see include/linux/module.h for more information
make[1]: Leaving directory '/home/cesar/Downloads/linux-source-4.15.0'
root@cesar-VirtualBox:/home/cesar/Documentos/programas/hello1#
```

Figura 2 - Print mostrando o comando make para a compilação do primeiro programa

Após a compilação deverá aparecer um arquivo com o final '.ko'. Este arquivo é extremamente importante e será usado na continuação da compilação .

Próximo passo é digitar o comando '`sudo insmod hello-1.ko`'. É bem provável que não mostre nenhuma mensagem no terminal. Para visualizar a mensagem do módulo, abrimos outro terminal e utilizamos o comando '`tail -f /var/log/kern.log`' ou '`journalctl --since "1 hour ago" | grep kernel`' para podemos ver se a compilação foi correta. As imagens abaixo mostra que a compilação foi feita com sucesso.

```
root@cesar-VirtualBox: /home/cesar/Documentos/programas/hello1
root@cesar-VirtualBox:/home/cesar/Documentos/programas/hello1# sudo insmod hello
-1.ko
root@cesar-VirtualBox:/home/cesar/Documentos/programas/hello1#
```

Figura 3 - Print mostrando o comando insmod

```
cesar@cesar-VirtualBox: /
cesar@cesar-VirtualBox:/$ tail -f /var/log/kern.log
Aug 30 15:27:16 cesar-VirtualBox gnome-session-binary[1277]: Entering running st
ate
Aug 30 16:13:44 cesar-VirtualBox kernel: [ 2022.547017] hello_1: loading out of
tree module taints kernel.
Aug 30 16:13:44 cesar-VirtualBox kernel: [ 2022.547020] hello_1: module license
'unspecified' taints kernel.
Aug 30 16:13:44 cesar-VirtualBox kernel: [ 2022.547020] Disabling lock debugging
due to kernel taint
Aug 30 16:13:44 cesar-VirtualBox kernel: [ 2022.547040] hello_1: module verifica
tion failed: signature and/or required key missing - tainting kernel
Aug 30 16:13:44 cesar-VirtualBox kernel: [ 2022.548666] Hello world 1.
Aug 30 16:17:10 cesar-VirtualBox kernel: [ 3047.423010] Goodbye world 1.
Aug 30 16:17:23 cesar-VirtualBox kernel: [ 3051.072116] Hello world 1.
Aug 30 16:17:29 cesar-VirtualBox kernel: [ 3057.074220] Goodbye world 1.
Aug 30 16:17:37 cesar-VirtualBox kernel: [ 3065.190533] Hello world 1.
```

Figura 4 - Print mostrando o resultado da compilação do primeiro módulo

Após a compilação, o módulo pode ser retirado do kernel com o comando '`sudo rmmod hello_1`'.

Os passos para a compilação dos próximos programas serão os mesmos portanto, as explicações sobre os comandos será reduzida, focando mais nos resultados.

Segundo módulo: hello-2

A imagem abaixo mostra o arquivo 'Makefile' usado para a compilação do módulo:

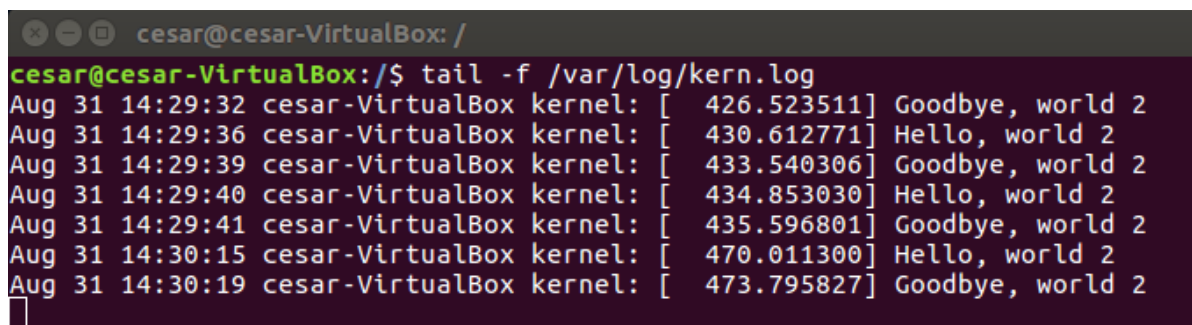
```
obj-m += hello-2.o

all:
    make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules

clean:
    make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

Figura 5 - Arquivo Makefile do segundo módulo

A imagem a seguir mostra o resultado da compilação do módulo.

A terminal window titled 'cesar@cesar-VirtualBox: /' shows the command 'tail -f /var/log/kern.log' being executed. The output displays a series of kernel messages from 'cesar-VirtualBox' with timestamps and hexadecimal addresses. The messages alternate between 'Hello, world 2' and 'Goodbye, world 2'.

```
cesar@cesar-VirtualBox:/$ tail -f /var/log/kern.log
Aug 31 14:29:32 cesar-VirtualBox kernel: [ 426.523511] Goodbye, world 2
Aug 31 14:29:36 cesar-VirtualBox kernel: [ 430.612771] Hello, world 2
Aug 31 14:29:39 cesar-VirtualBox kernel: [ 433.540306] Goodbye, world 2
Aug 31 14:29:40 cesar-VirtualBox kernel: [ 434.853030] Hello, world 2
Aug 31 14:29:41 cesar-VirtualBox kernel: [ 435.596801] Goodbye, world 2
Aug 31 14:30:15 cesar-VirtualBox kernel: [ 470.011300] Hello, world 2
Aug 31 14:30:19 cesar-VirtualBox kernel: [ 473.795827] Goodbye, world 2
```

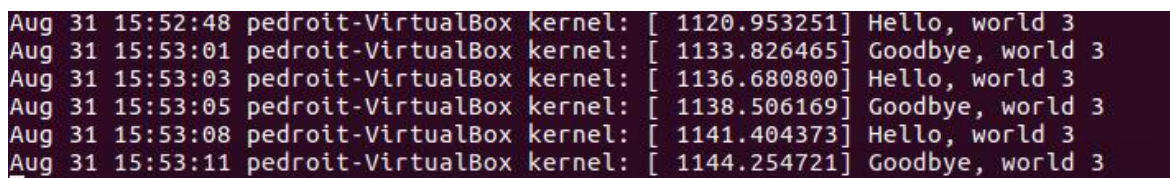
Figura 6 - Print mostrando o resultado da compilação do segundo módulo

Cada 'Hello, world 2' é mostrado quando o módulo é inserido no kernel e cada 'Goodbye world 2' é mostrado quando o módulo é retirado do kernel.

Terceiro módulo: hello-3

Para o arquivo 'Makefile', basta mudar a primeira linha, substituindo o nome do arquivo pelo desejado.

A imagem a seguir mostra o resultado da compilação do módulo.

A terminal window shows the output of 'tail -f /var/log/kern.log' for the third module. The messages alternate between 'Hello, world 3' and 'Goodbye, world 3' with timestamps and hexadecimal addresses.

```
Aug 31 15:52:48 pedroit-VirtualBox kernel: [ 1120.953251] Hello, world 3
Aug 31 15:53:01 pedroit-VirtualBox kernel: [ 1133.826465] Goodbye, world 3
Aug 31 15:53:03 pedroit-VirtualBox kernel: [ 1136.680800] Hello, world 3
Aug 31 15:53:05 pedroit-VirtualBox kernel: [ 1138.506169] Goodbye, world 3
Aug 31 15:53:08 pedroit-VirtualBox kernel: [ 1141.404373] Hello, world 3
Aug 31 15:53:11 pedroit-VirtualBox kernel: [ 1144.254721] Goodbye, world 3
```

Figura 7 - Print mostrando o resultado da compilação do terceiro módulo

Quarto módulo: hello-4.c

A imagem a seguir mostra o resultado da compilação do módulo.

```
Aug 31 15:50:12 pedroit-VirtualBox kernel: [ 965.514487] Hello, world 4
Aug 31 15:50:23 pedroit-VirtualBox kernel: [ 976.245834] Goodbye, world 4
Aug 31 15:50:25 pedroit-VirtualBox kernel: [ 978.108837] Hello, world 4
Aug 31 15:50:26 pedroit-VirtualBox kernel: [ 979.534274] Goodbye, world 4
Aug 31 15:50:30 pedroit-VirtualBox kernel: [ 983.069719] Hello, world 4
Aug 31 15:50:32 pedroit-VirtualBox kernel: [ 984.895368] Goodbye, world 4
```

Figura 8 - Print da tela com o resultado da compilação do quarto módulo

Quinto módulo: hello-5

Neste módulo podemos passar diferentes parâmetros para sua compilação. Como será mostrado nas imagens abaixo, conforme alteramos os parâmetros alteramos o resultado da compilação.

```
root@cesar-VirtualBox: /home/cesar/Documentos/programas/hello5
root@cesar-VirtualBox:/home/cesar/Documentos/programas/hello5# sudo insmod hello
-5.ko myshort="2" myint="255" mylong="87899"
```

Figura 9 - Compilação do módulo 5 com diferentes parâmetros

```
Sep  1 13:13:39 cesar-VirtualBox kernel: [ 749.344970] Hello, world 5
Sep  1 13:13:39 cesar-VirtualBox kernel: [ 749.344970] =====
Sep  1 13:13:39 cesar-VirtualBox kernel: [ 749.344972] myshort is a short integer: 2
Sep  1 13:13:39 cesar-VirtualBox kernel: [ 749.344973] myint is an integer: 255
Sep  1 13:13:39 cesar-VirtualBox kernel: [ 749.344973] mylong is a long integer: 87899
Sep  1 13:13:39 cesar-VirtualBox kernel: [ 749.344974] mystring is a string: blah
Sep  1 13:13:39 cesar-VirtualBox kernel: [ 749.344974] myintArray[0] = -1
Sep  1 13:13:39 cesar-VirtualBox kernel: [ 749.344975] myintArray[1] = -1
Sep  1 13:13:39 cesar-VirtualBox kernel: [ 749.344976] got 0 arguments for myintArray.
```

Figura 10 - Print mostrando um dos resultados da compilação do quinto módulo

```
Aug 31 15:56:26 pedroit-VirtualBox kernel: [ 1339.669160] Hello, world 5
Aug 31 15:56:26 pedroit-VirtualBox kernel: [ 1339.669160] =====
Aug 31 15:56:26 pedroit-VirtualBox kernel: [ 1339.669161] myshort is a short integer: 1
Aug 31 15:56:26 pedroit-VirtualBox kernel: [ 1339.669162] myint is an integer: 420
Aug 31 15:56:26 pedroit-VirtualBox kernel: [ 1339.669163] mylong is a long integer: 9999
Aug 31 15:56:26 pedroit-VirtualBox kernel: [ 1339.669164] mystring is a string: blah
Aug 31 15:56:26 pedroit-VirtualBox kernel: [ 1339.669165] myintArray[0] = -1
Aug 31 15:56:26 pedroit-VirtualBox kernel: [ 1339.669166] myintArray[1] = -1
Aug 31 15:56:26 pedroit-VirtualBox kernel: [ 1339.669167] got 0 arguments for myintArray.
Aug 31 15:57:38 pedroit-VirtualBox kernel: [ 1411.419196] Goodbye, world 5
Aug 31 15:59:48 pedroit-VirtualBox kernel: [ 1540.814855] Hello, world 5
Aug 31 15:59:48 pedroit-VirtualBox kernel: [ 1540.814855] =====
Aug 31 15:59:48 pedroit-VirtualBox kernel: [ 1540.814856] myshort is a short integer: 1
Aug 31 15:59:48 pedroit-VirtualBox kernel: [ 1540.814857] myint is an integer: 420
Aug 31 15:59:48 pedroit-VirtualBox kernel: [ 1540.814858] mylong is a long integer: 9999
Aug 31 15:59:48 pedroit-VirtualBox kernel: [ 1540.814859] mystring is a string: blah
Aug 31 15:59:48 pedroit-VirtualBox kernel: [ 1540.814860] myintArray[0] = 10
Aug 31 15:59:48 pedroit-VirtualBox kernel: [ 1540.814861] myintArray[1] = 5
Aug 31 15:59:48 pedroit-VirtualBox kernel: [ 1540.814862] got 0 arguments for myintArray.
Aug 31 16:00:01 pedroit-VirtualBox kernel: [ 1553.993448] Goodbye, world 5
```

Figura 11 - Print mostrando um dos resultados da compilação do quinto módulo

Sexto módulo: startstop

Neste módulo, podemos ver como funciona a compilação de um módulo, quando este é dividido em dois arquivos diferentes. O primeiro arquivo (start.c) contém apenas a função de inicializar o módulo e já o segundo (stop.c) contém a função de saída do módulo. As imagens abaixo mostram o conteúdo de cada arquivo citado.

```
/*
 * start.c - Illustration of multi filed modules
 */

#include <linux/kernel.h>      /* We're doing kernel work */
#include <linux/module.h>      /* Specifically, a module */

int init_module(void)
{
    pr_info("Hello, world - this is the kernel speaking\n");
    return 0;
}
```

Figura 12 - Programa start.c

```
/*
 * stop.c - Illustration of multi filed modules
 */

#include <linux/kernel.h>      /* We're doing kernel work */
#include <linux/module.h>      /* Specifically, a module */

void cleanup_module()
{
    pr_info("Short is the life of a kernel module\n");
}
```

Figura 13 – Programa stop.c

Pelo fato do módulo ser composto por dois arquivos diferentes, o 'Makefile' deste também sofrerá mudanças.

```
obj-m += startstop.o
startstop-objs := start.o stop.o

all:
    make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules

clean:
    make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

Figura 14 - Arquivo Makefile do sexto módulo

Na primeira linha do arquivo, escolhemos um nome para o módulo e na segunda quais os arquivos que o compõe.

Para a compilação do módulo basta digitar 'sudo insmod (nome do módulo escolhido)' e para remove-lo do kernel 'sudo rmmod (nome do módulo escolhido)'. Portanto, o processo de compilação de um módulo com mais de um arquivo origem é bem simples, com pequenas alterações no arquivo 'Makefile'.

As imagens abaixo mostram o resultado da compilação deste módulo.

```
Sep  1 13:26:55 cesar-VirtualBox kernel: [ 1545.461970] Hello, world - this is the kernel speaking
Sep  1 13:27:05 cesar-VirtualBox kernel: [ 1555.815802] Short is the life of a kernel module
```

Figura 15 - Print mostrando o resultado da compilação do sexto módulo

```
Sep  1 14:43:36 beatriz kernel: [ 4553.476925] Hello, world - this is the kernel speaking
Sep  1 15:11:47 beatriz kernel: [ 6244.321842] Hello, world - this is the kernel speaking
Sep  1 15:15:11 beatriz kernel: [ 6448.748224] Short is the life of a kernel module
```

Figura 16 - Print mostrando o resultado da compilação do sexto módulo

Sétimo módulo: chardev

A imagem abaixo mostra o resultado da compilação do módulo.

```
Sep  1 21:51:36 beatriz kernel: [ 522.078176] I was assigned major number 240.
Sep  1 21:51:36 beatriz kernel: [ 522.078268] Device created on /dev/chardev
```

Figura 17 - Print dos comandos para compilação do sétimo módulo

Oitavo módulo: procfs1

As imagens abaixo mostram o resultado da compilação do módulo.

```
root@cesar-VirtualBox: /home/cesar/Documentos/programas/procfs1
root@cesar-VirtualBox:/home/cesar/Documentos/programas/procfs1# sudo insmod procfs1.ko
root@cesar-VirtualBox:/home/cesar/Documentos/programas/procfs1# sudo rmmod procfs1
root@cesar-VirtualBox:/home/cesar/Documentos/programas/procfs1#
```

Figura 18 - Print dos comandos para compilação do oitavo módulo

```
Sep  1 19:57:59 cesar-VirtualBox kernel: [ 1147.393875] /proc/helloworld created
Sep  1 20:00:05 cesar-VirtualBox kernel: [ 1273.206074] /proc/helloworld removed
Sep  1 20:00:24 cesar-VirtualBox kernel: [ 1291.944662] /proc/helloworld created
Sep  1 20:00:46 cesar-VirtualBox kernel: [ 1313.699190] procfile read helloworld
Sep  1 20:01:10 cesar-VirtualBox kernel: [ 1338.638391] /proc/helloworld removed
Sep  1 20:01:56 cesar-VirtualBox kernel: [ 1384.521332] /proc/helloworld created
Sep  1 20:02:27 cesar-VirtualBox kernel: [ 1415.214488] procfile read helloworld
Sep  1 20:02:41 cesar-VirtualBox kernel: [ 1429.379768] procfile read helloworld
```

Figura 19 - Print mostrando o resultado da compilação do oitavo módulo

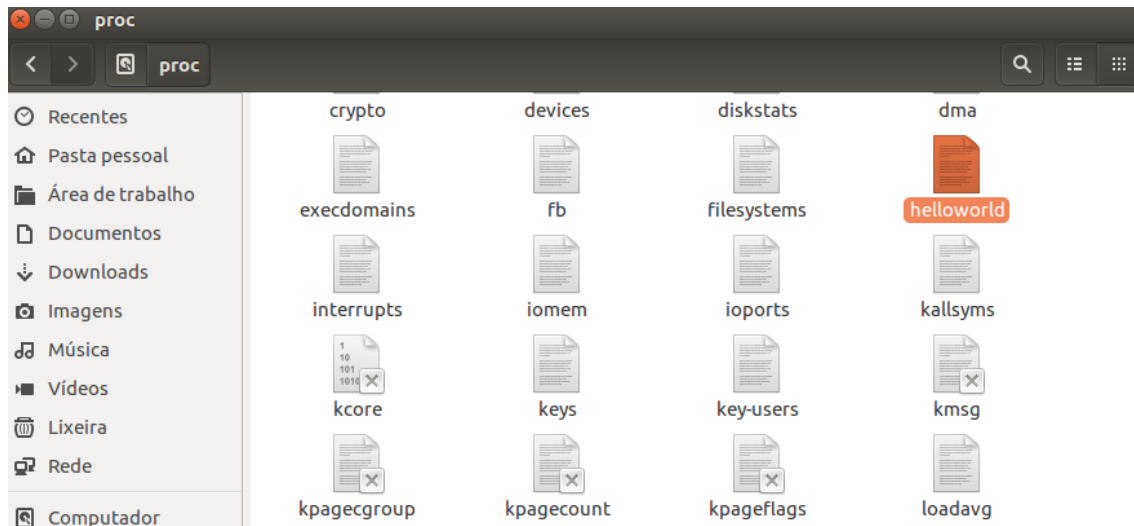


Figura 20 - Print mostrando o arquivo criado pelo módulo

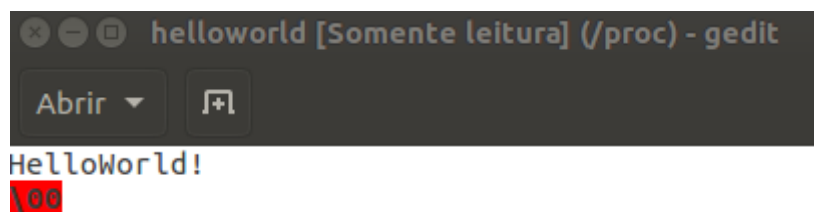


Figura 21 - Print mostrando o conteúdo do arquivo criado pelo módulo

Nono módulo: procfs2

As imagens abaixo mostram o resultado da compilação do módulo.

```
Sep 1 20:13:19 cesar-VirtualBox kernel: [ 2067.142047] Hello world 1.
Sep 1 20:13:27 cesar-VirtualBox kernel: [ 2074.707749] Goodbye world 1.
Sep 1 20:15:07 cesar-VirtualBox kernel: [ 2175.421102] /proc/buffer1k created
Sep 1 20:15:55 cesar-VirtualBox kernel: [ 2223.355099] procfile read buffer1k
Sep 1 20:16:38 cesar-VirtualBox kernel: [ 2266.236014] /proc/buffer1k removed
```

Figura 22 - Print mostrando o resultado da compilação do nono módulo

```
fabricio@fabricio-VirtualBox:/develop/kernel/hello-1$ journalctl --since "1 hour
ago" | grep kernel
Ago 31 16:47:06 fabricio-VirtualBox sudo[3891]: fabricio : TTY=pts/4 ; PWD=/deve
lop/kernel/hello-1 ; USER=root ; COMMAND=/sbin/insmod helloworld.ko
Ago 31 16:47:06 fabricio-VirtualBox kernel: /proc/buffer1k created
Ago 31 16:47:11 fabricio-VirtualBox sudo[3894]: fabricio : TTY=pts/4 ; PWD=/deve
lop/kernel/hello-1 ; USER=root ; COMMAND=/sbin/lsmmod
```

Figura 23 - Print mostrando o resultado da compilação do nono módulo

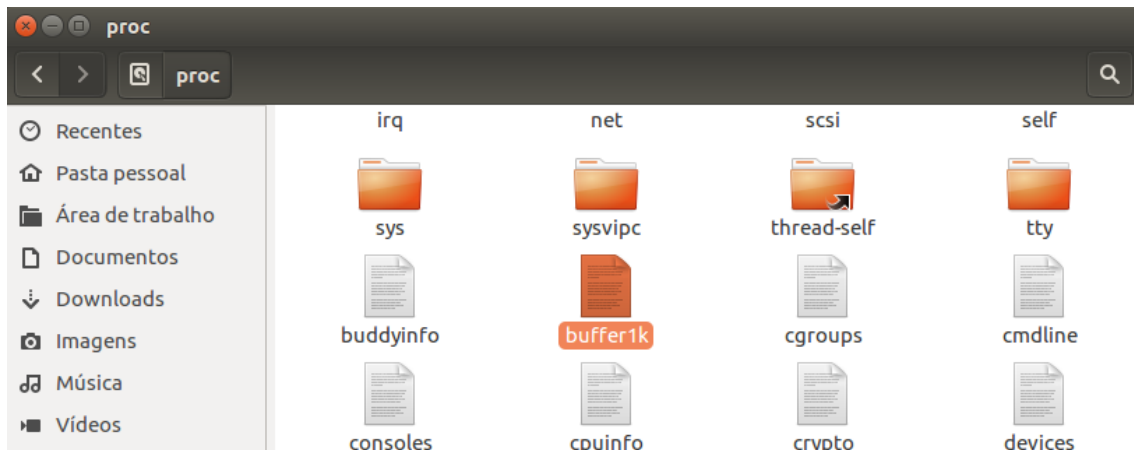


Figura 24 - Print mostrando o arquivo criado pelo módulo

Décimo módulo: procfs3

A imagem abaixo mostra o resultado da compilação do módulo.

```
Set 03 18:39:18 pedroit-VirtualBox kernel: /proc/buffer2k created
Set 03 18:39:29 pedroit-VirtualBox kernel: procfs_read: read 0 bytes
Set 03 18:39:34 pedroit-VirtualBox kernel: procfs_read: END
Set 03 18:39:34 pedroit-VirtualBox kernel: procfs_read: read 0 bytes
Set 03 18:39:34 pedroit-VirtualBox kernel: procfs_read: END
Set 03 18:39:34 pedroit-VirtualBox kernel: procfs_read: read 0 bytes
Set 03 18:42:23 pedroit-VirtualBox kernel: /proc/buffer2k removed
```

Figura 25 - Print mostrando o resultado da compilação do décimo módulo

Décimo primeiro módulo: procfs4

A imagem abaixo mostra o resultado da compilação do módulo.

```
Set 03 18:42:23 pedroit-VirtualBox kernel: /proc/buffer2k removed
Set 03 18:48:17 pedroit-VirtualBox kernel: /proc/iter removed
Set 03 18:48:42 pedroit-VirtualBox kernel: /proc/helloworld created
```

Figura 26 - Print mostrando o resultado da compilação do décimo primeiro módulo

Conclusão

Com o experimento foi possível compreender todos os passos da implementação, compilação e instalação de um módulo para o kernel Linux. Mesmo que as utilidades de cada módulo não fossem práticas, foi importante entender a diferença da programação em espaço de kernel, uma vez que as bibliotecas são diferentes e o jeito de programar também. Também foi compreendido o uso de alguns comandos para compilação como: 'insmod', 'rmmod', 'journalctl' e 'tail -f'.