Alteração do kernel do Linux para prover suporte a sistemas de tempo real.

Kevin Willyn e Matheus Naranjo

UFRR - Universidade Federal de Roraima

RESUMO

Demonstrar, descrever e compreender o funcionamento de um kernel do Linux, seus módulos de uso assim como se desenvolve um módulo para o seguinte sistema operacional, demonstrando por meio de um singelo passo a passo, visando a alteração do kernel para funcionamento em tempo real por meio do módulo RTAI – REAL TIME APPLICATION INTERFACE FOR LINUX.

PALAVRAS-CHAVE

Kernel; Linux; Módulos; RTAI.

INTRODUÇÃO

Alterar o kernel do Linux apresenta-se como um desafio em um primeiro momento. A instalação de um módulo que permita que o trabalhar de forma em tempo real, com diminuição de latências para algumas aplicações especificas e funcionamentos adequados para determinadas atividades. O seguinte artigo visa explicar o funcionamento de instalação dessa ferramenta assim como descrever o funcionamento do kernel do Linux em si. Como trabalha, quais são os módulos e como encontrá-los no sistema operacional e até mesmo como desenvolver o próprio por meio de um exemplo simples.

1 - Descrição do funcionamento do kernel do Linux, bem como suas respectivas divisões de suporte ao sistema operacional

Antes de iniciar a modificação do kernel do Linux, é necessário compreender como ele funciona. O kernel é o principal componente do sistema operacional e, também é a interface central entre o hardware e os processos executados por um computador. Por meio dele, podemos estabelecer comunicações entre recursos do sistema, além de gerenciar os mesmos com melhor eficiência.

Conforme o site do Red Hat define:

> "Pense da seguinte forma: o kernel é como um assistente pessoal muito ocupado que trabalha para um executivo poderoso (o hardware). É trabalho do assistente transmitir as mensagens e solicitações (processos) dos funcionários e do público (usuários) para o executivo, lembrar o que está armazenado e onde (memória), e determinar quem tem acesso ao executivo, em que horário e por quanto tempo".

O Kernel possui funções importantes, que se dividem em **quatro**. São elas:

Gerenciamento da Memória: é dividida em dois componentes: o responsável por alocação e liberação de memória física e o responsável pela memória virtual. Podemos monitorar o uso de memória usado em armazenamento de arquivos, dados etc., além do local onde são armazenados. No kernel Linux, as alocações são feitas de forma estática por drivers que separam determinado espaço de memória logo no processo de inicialização do sistema. Temos que o sistema de memória virtual, o kmalloc (alocador de comprimento de variável) e os caches, são os mais importantes sistemas de alocação do Linux.

Gerenciamento de Processos: determina quais processos podem ter acesso a CPU (Unidade Central de Processamento), assim como definir o período de duração e quando podem. Deve-se levar em conta a ideia de que cada processador só executa uma tarefa por vez, e por meio do **TIME SHARING** podemos ir rotacionando o uso dos processos nas CPUs disponíveis, definindo tempos de uso em cada. No

que tange aos tipos de processo, temos dois: os convencionais, que geralmente são voltados a uso do sistema no geral e os de tempo real, que são executados imediatamente, sendo em sua maioria, processos internos do kernel do Linux. Identificam-se pela sigla "rt" no código, uma referência a *Real Time.*

Drivers de dispositivos: nada mais são do que pontes entre dispositivos a serem usados na máquina e o kernel. Obviamente, dependendo de cada distribuição Linux usada, teremos níveis de suporte para dispositivos de entrada e saída. Para alguns, nenhum suporte, com em casos de drivers de rede. Em outros, suporte mínimo, e em alguns casos, suporte estendido.

Chamadas do sistema e segurança: aqui, recebem-se solicitações dos processos que irão executar certos serviços. Aqui, encontramos o termo SCI – Implementações de chamadas do sistema. Com ele, define-se a arquitetura e a forma como se usam as chamadas. Abaixo, temos um exemplo de como funcionam as chamadas de sistema no Linux.

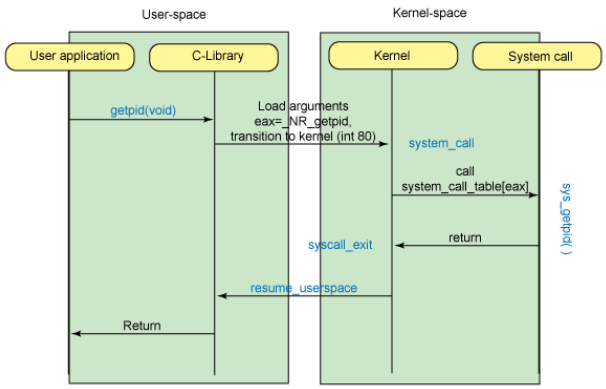


Figura 1- Fluxo de Exemplo de chamada do Sistema

No que tange ao funcionamento de um Kernel Linux, deve-se compreender que há duas formas de implementação no sistema operacional, a depender é claro, de cada um. Temos os Kernel monolítico, que é estruturado como arquivo binário, do qual possui desempenho superior a passagem de mensagens, mas com inúmeras desvantagens no que tange a desperdício de recurso. Há também o microkernel. Este, por sua vez, fica com apenas uma parte do seu núcleo executando de forma protegida para acesso de hardware, assim como comunicação entre processos e afins. O resto de seu kernel, executa voltado diretamente ao usuário, mas sem que esse necessite realizar os vários passos complexos para realizar uma manipulação simples. Na prática, o modo monolítico é superior em desempenho.

Recursos são acessados por protocolos client/server padrão, e isso afeta a forma como ocorrem as comunicações entre processos sobre um mesmo processador. O módulo que comunica processos no geral, tem funções relacionadas a receber mensagens provenientes de processadores diferentes e deles, enviar para os outros processadores de destino. Aqui, os clientes são os programas de aplicação, e os servidores, os serviços do sistema.

No geral, tem-se definido que o Kernel do Linux é de núcleo monolítico, de código aberto para sistemas operacionais Unix.

2 - Módulos do Kernel do Linux

Seu design é modular, quando ocorre inicialização do sistema operacional, carrega-se o kernel residente na memória, que é o kernel de "tamanho mínimo" para funcionamento. Conforme se chamam outros módulos que não estão presentes no kernel residente, o módulo do Kernel os vai dinamicamente carregando na memória.

O próprio sistema decide quais módulos necessitam ser carregados a partir da verificação de hardware. O programa configura o mecanismo de carregamento dinâmico para trabalhar melhor.

Atenta-se ao fato de que caso seja adicionado um hardware novo a máquina, caso este após a instalação requeira módulo do kernel, o sistema deve ser configurado para carregar o módulo do kernel apropriado a nova peça. Entra em ação o programa **KUDZU,** que detectará a nova peça caso seja possível e ainda configura seu módulo. Pode-se ainda fazer isso manualmente pelo comando:

```
/etc/modules.conf
```

Para saber quais módulos estão rodando no exato momento, utiliza-se o comando:

```
lsmod
```

O pacote MODUTILS é o responsável por administrar os módulos presentes no Linux. Com ele podemos saber se um módulo foi ou não carregado com sucesso.

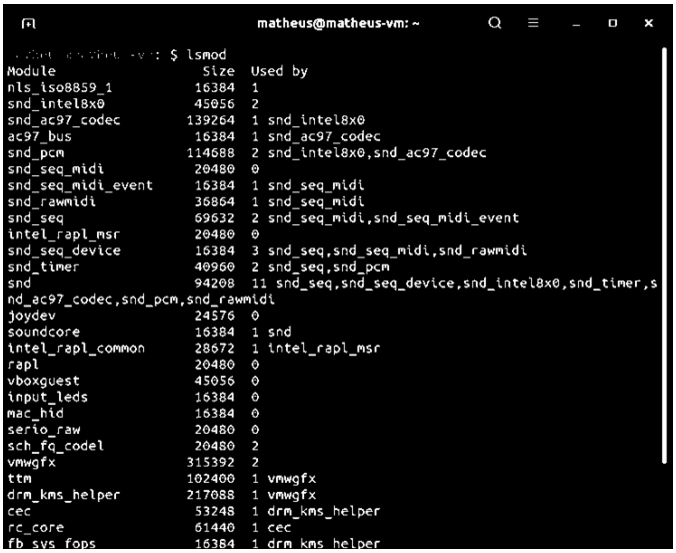


Figura 2 - Exemplo de saída gerada para saber os módulos usados no momento

Podemos ver o nome do módulo usado, o tamanho da memória usada por ele, além de ver quem o está usando. Podemos criar módulos por meio de *depmod*, além do *insmod* para inserir um módulo no kernel Linux, *modinfo* para saber de informações sobre um módulo. Por fim, temos o *modprobe* que adiciona e remove módulos do kernel e o *rmmod* que é um simples programa que remove um módulo do kernel.

Podemos listar os módulos built-in presentes no kernel por meio do comando:

```
more /lib/modules/$(uname -r)/modules.builtin |
                    head -10
```

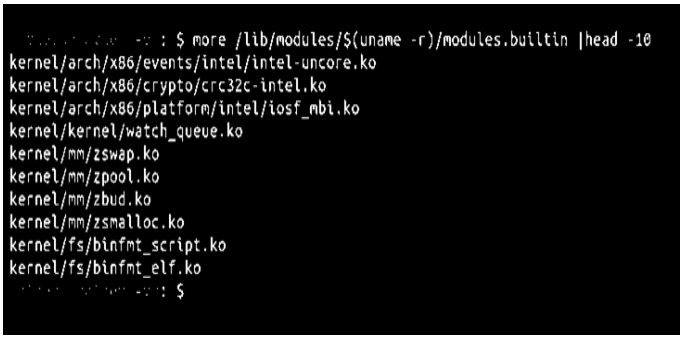


Figura 3- Resultado exibindo modulos built-in do kernel

3 – Como desenvolver um módulo para Kernel do Linux

Conforme cita Robert W. Oliver II (2017), escrever um módulo para kernel Linux é algo que se der errado, pode comprometer todo o sistema. Quando se insere um código diretamente no kernel Linux, ele roda no nível zero de acesso, o que significa que podemos ter acesso a último nível de proteção que há. Qualquer erro fica sobre conta e risco daquele que está desenvolvendo algo. Há um acesso completo ao sistema.

Para começar a desenvolver algo, aplicamos primeiramente o seguinte o código:

```
apt-get install build-essential Linux-headers-
                    `uname -r´
```

Com ele baixamos as ferramentas essenciais para inicializar nosso código. Há nele, os headers que serão inseridos no código.

Preparamos o local onde iremos armazenar o arquivo, podendo ser assim:

```
mkdir ~/src/lkm_example

cd ~/src/lkm_example
```

E assim, cria-se o código. No caso, chamaremos os arquivos de *lkm_example.c*:

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

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Kevin Willyn and Matheus
Naranjo");
MODULE_DESCRIPTION("A simple example Linux
module for UFRR.");
MODULE_VERSION("0.01");

static int __init lkm_example_init(void) {
 printk(KERN_INFO "Hello, World! We are at
College\n");
 return 0;
}
static void __exit lkm_example_exit(void) {
 printk(KERN_INFO "Goodbye, World!\n");
}
module_init(lkm_example_init);
module_exit(lkm_example_exit);
```

O código acima inclui os cabeçalhos necessários para o desenvolvimento para arquivos de kernel. MODULE_LICENSE é o conjunto que permite acesso de licença do módulo. INIT e EXIT são as funções estáticas que retornam inteiros, sendo o primeiro para o carregamento e o segundo para quando finalizamos sua execução. Para printar saídas no kernel, no lugar da função printf, usada na linguagem C, usamos o printk. Ele não possui os mesmos parâmetros, tendo funcionamento diferente do printf. Ainda possuímos as funções MODULE_INIT e MODULE_EXIT que dizem ao kernel quais funções serão usadas para o carregamento e o descarregamento de funções no kernel.

```
linux-5.14.10/virt/kvm/async_pf.h
linux-5.14.10/virt/kvm/binary_stats.c
linux-5.14.10/virt/kvm/coalesced_mmio.c
linux-5.14.10/virt/kvm/coalesced_mmio.h
linux-5.14.10/virt/kvm/dirty_ring.c
linux-5.14.10/virt/kvm/eventfd.c
linux-5.14.10/virt/kvm/irqchip.c
linux-5.14.10/virt/kvm/kvm_main.c
linux-5.14.10/virt/kvm/mmu_lock.h
linux-5.14.10/virt/kvm/vfio.c
linux-5.14.10/virt/kvm/vfio.h
linux-5.14.10/virt/lib/
linux-5.14.10/virt/lib/Kconfig
linux-5.14.10/virt/lib/Makefile
linux-5.14.10/virt/lib/irqbypass.c
root@matheus-vm:/usr/src# tar xjf rtai-3.6.tar.bz2
tar (child): rtai-3.6.tar.bz2: Cannot open: No such file or directory
tar (child): Error is not recoverable: exiting now
tar: Child returned status 2
tar: Error is not recoverable: exiting now
root@matheus-vm:/usr/src# tar xjf /home/matheus/Downloads/rtai-3.6.tar.bz2
```

Figura 4 - Saída gerada

Por fim, necessita-se do markfile. Para sua criação, deve-se prestar muita atenção a tabulações, espaços em branco e afins.

```
obj-m += lkm_example.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- Saídas geradas

Se rodarmos “make”, teremos o módulo compilado com sucesso. Um arquivo “lkm_example.ko” será gerado. Podemos inserir o arquivo do módulo e testá-lo. Implementamos o seguinte código:

```
# sudo insmod lkm_example.ko
```

Mesmo tudo funcionando, não é possível ver o resultado, visto que printk não gera saída para o console. Para vermos a saída de verdade, aplicamos *sudo dmesg* para então ver a saída. Automatizando as coisas, ficamos assim:

```
test:
 sudo dmesg -C
 sudo insmod lkm_example.ko
 sudo rmmod lkm_example.ko
 dmesg
```

Rodando agora o Markfile, aplicando *make test*, chegamos ao mesmo resultado.

Esta forma de desenvolver um módulo, apesar de complicada em um primeiro momento, mostra-se na verdade como uma porta de possibilidades para que possamos desenvolver módulos mais interessantes e que gerem algum impacto real em aplicações ou até mesmo na forma como utilizamos o Linux.

Para poder então, transformar nosso sistema operacional em tempo real, devemos seguir os seguintes passos. Primeiramente, devemos, abrindo o terminal do Linux e adentrar o modo super usuário por meio do seguinte comando:

```
sudo -i
```

Colocando a senha (caso tenha), já temos acesso completo e irrestrito de nossa máquina. A seguir, deveremos fazer a criação dos diretórios dos quais serão instalados os arquivos a serem usados. Aplicando o comando mkdir */usr/src*, criamos a pasta da qual nosso projeto irá ser trabalhado. Dando prosseguimento, deveremos instalar as versões do RTAI a ser usado e do Kernel do Linux puro. O motivo é simples, aplicando em um kernel limpo, podemos modificá-lo sem maiores problemas de afetarmos o nosso próprio kernel. Para o projeto, estamos usando o RTAI versão 5.3 e o kernel do Linux versão 4.19.177.

O próximo passo é instalar o Linux e o RTAI. Clicando no link você será levado diretamente para o site o qual você instalará o kernel e o segundo link para o RTAI. Baixando as versões com final “.tar.bz2” por exemplo, fará com que ao descompactar, tenha de aplicar alguns comandos. Abaixo, um exemplo visual do download.

```
root@matheus-vm:/usr/src# wget www.rati.org/RTAI/rtai-3.6.bz2
--2021-10-07 16:20:13--  http://www.rati.org/RTAI/rtai-3.6.bz2
Resolving www.rati.org (www.rati.org)... 34.102.136.180
Connecting to www.rati.org (www.rati.org)|34.102.136.180|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 2548 (2,5K) [text/html]
Saving to: ‘rtai-3.6.bz2’

rtai-3.6.bz2       100%[==============>]   2,49K  --.-KB/s    in 0s

2021-10-07 16:20:14 (215 MB/s) - ‘rtai-3.6.bz2’ saved [2548/2548]
```

Figura 6 - Instalação do RTAI

Baixado os arquivos, os descompacte no diretório “src”. Passe o caminho completo de onde estão os arquivos e então aplique o seguinte comando como no exemplo:

```
    # tar xjf rtai-5.3.tar.bz2

    # tar xjf linux-4.19.177.tar.bz2
```

A imagem acima ilustra o processo de descompactação dos arquivos.

Indo na pasta onde está o kernel do Linux descompactado, aplique o seguinte comando:

```
# patch -p1 <
../rtai/base/arch/i386/patches/hal-linux-
2.6.23-i386-1.12-00.patch
```

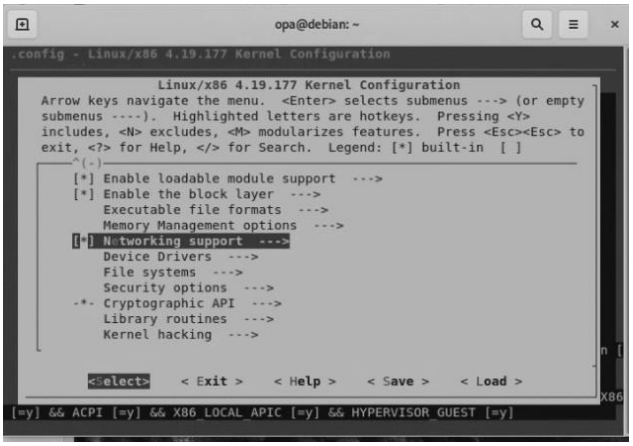
Após isso, copiamos a configuração de nosso kernel o qual nos fará criar a oportunidade de modificar o kernel puro. # make menuconfig

```
#  cp /boot/config-`uname -r` .config
```

E agora, digite o seguinte:

```
# make menuconfig
```

E se abrirá a seguinte tela:



Lendo a documentação, você terá acesso mais qualificado e rico em detalhes sobre quais configurações deve manter marcado com “sim” e quais deve estar como “não”. Após isso, você deverá compilar o kernel.

```
#      make-kpkg --initrd kernel_image kernel_headers
```

Após a compilação, volte a pasta /src e crie a pasta de firmware. Alguns procedimentos dependerão da versão de Kernel e RTAI usados, assim como das distribuições Linux usadas.

```
# mkdir /lib/firmware/2.6.23-rtai
```

Após isso, instale os pacotes Debian criados.

```
# dpkg -i *.deb
```

E reinicie com o kernel novo.

Por fim, compile o RTAI. Apague qualquer versão anterior caso exista e vá para a pasta do RTAI dentro do /src. Aplique:

```
# make menuconfig
```

Compile e instale o RTAI e depois faça reboot.

```
# make
# make install
# reboot
```

Por fim, teste

```
# cd /usr/realtime/testsuit/user

# cd latency
# ./run
# cd ../preempt
# ./run+
# cd ../switch
# ./run
```

O teste nos indicará se a latência está baixa, e se a troca de informações entre os processos e afins está sendo ocorrendo dentro de um certo tempo.

CONCLUSÃO

A modificação, apesar de trabalhosa, apresenta resultado satisfatório. Em muitas etapas, é possível notar que passos precisarão ser refeitos com bastante cuidado, uma vez que poderão afetar o desempenho final de compilação. Contudo, apesar da dificuldade de instalação e configuração, demonstra-se satisfatória para os propósitos apresentados.

REFERÊNCIAS BIBLIOGRÁFICAS

FROHLICH, A. A.; CARMINATI, ; RIBEIRO, H. R. usfc. **linhas.usfc**, 2008. Disponivel em: <https://lisha.ufsc.br/teaching/os/ine5355-2008-2/work/rtai.pdf>. Acesso em: 3 out. 2021.

II, R. W. O. blog.sourcerer.io. **blog.sourcerer.io**, 2017. Disponivel em: <https://blog.sourcerer.io/writing-a-simple-linux-kernel-module-d9dc3762c234>. Acesso em: 28 set. 2021.

LAGES,. ufrgs. **ece.ufrgs**, 2014. Disponivel em: <http://www.ece.ufrgs.br/~fetter/ele213/rtai_install.pdf>. Acesso em: 17 Setembro 2021.

RED HAT. red hat. **redhat.com**, -. Disponivel em: <https://www.redhat.com/pt-br/topics/linux/what-is-the-linux-kernel>. Acesso em: 27 set. 2021.

RTAI.ORG. **rtai**, 2009. Disponivel em: <https://www.rtai.org/userfiles/downloads/RTAI/>. Acesso em: 2 out. 2021.