

UNIVERSIDADE FEDERAL DE RORAIMA

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

Alunos: Luis Henrique e João Lucas

Boa Vista  
2020

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

Existem diferentes maneiras e considerações arquitetônicas ao construir um kernel do zero. A maioria dos kernels se enquadra em um dos três tipos: monolítico, microkernel e híbrido. O Linux é um kernel monolítico, enquanto o OS X (XNU) e o Windows 7 usam kernels híbridos.

Os kernels **monolíticos** são o oposto dos microkernels porque abrangem não apenas a CPU, a memória e o IPC, mas também incluem coisas como drivers de dispositivo, gerenciamento do sistema de arquivos e chamadas do servidor do sistema. Kernels monolíticos tendem a ser melhores no acesso a hardware e multitarefa porque se um programa precisa obter informações da memória ou outro processo em execução, ele tem uma linha mais direta para acessá-lo e não precisa esperar em uma fila para fazer as coisas.

Um **microkernel** age de forma à gerenciar apenas o que é necessário: CPU, memória e IPC. Quase todo o resto no computador pode ser visto como um acessório a ser manipulado no modo de usuário. Sua grande vantagem é portabilidade pois não é necessário se preocupar após trocar uma placa de vídeo ou mesmo o sistema operacional, desde que este ainda tente acessar o hardware da mesma forma. Os microkernels também ocupam muito pouco espaço, tanto de memória quanto de de instalação, e tendem a ser mais seguros porque apenas processos específicos são executados no modo de usuário que não tem permissões altas como no modo supervisor.

Kernels **híbridos** têm a capacidade de escolher se desejam executar algo no modo de usuário ou no modo supervisor. Algo como drivers de dispositivos e E / S do sistema de arquivos serão executados no modo de usuário, enquanto o IPC e as chamadas do servidor ficam no modo supervisor. Isso oferece o melhor dos dois mundos, mas geralmente exigirá mais trabalho do fabricante do hardware porque toda a responsabilidade do driver é dela. Ele também pode ter alguns dos problemas de latência inerentes aos microkernels.

Por ser monolítico, o kernel do linux tem maior complexidade sobre os outros tipos de kernels. Este foi um recurso de design que estava sob muito debate nos primeiros dias do Linux e ainda carrega algumas das velhas falhas de design que os kernels monolíticos costumam ter.

Uma ação que os desenvolvedores do kernel Linux tomaram para contornar essas falhas foi criar módulos do kernel que pudessem ser carregados e descarregados em tempo de execução, o que significa que o usuário pode adicionar ou remover recursos do kernel na hora. Mais do que apenas adicionar funcionalidade de hardware ao kernel, é possível incluir módulos que executam processos de servidor, como virtualização de baixo nível e também permitir que todo o kernel seja substituído sem a necessidade de reiniciar o computador em alguns casos.

### Módulos do Kernel

Módulos de kernel, também conhecidos como módulo de kernel carregável (LKM), são essenciais para mantê-lo funcionando com todo o seu hardware sem consumir toda a memória disponível.

Um módulo normalmente adiciona funcionalidade ao kernel base para coisas como dispositivos, sistemas de arquivos e chamadas de sistema. Os LKMs têm a extensão de arquivo .ko e são normalmente armazenados no diretório / lib / modules. Por causa de sua natureza modular, você pode facilmente personalizar seu kernel configurando módulos para carregar, ou não carregar,

durante a inicialização com o comando `menuconfig` ou editando seu arquivo `/boot/config`, ou você pode carregar e descarregar módulos rapidamente com o `modprobe` comando.

Módulos de terceiros e de código fechado estão disponíveis em algumas distribuições, como o Ubuntu, e podem não ser instalados por padrão porque o código-fonte dos módulos não está disponível. O desenvolvedor do software (ou seja, nVidia, ATI, entre outros) não fornece o código-fonte, mas sim constrói seus próprios módulos e compila os arquivos `.ko` necessários para distribuição.

### Módulos do Kernel no Linux

O kernel do Linux tem um design modular. No momento da inicialização, somente um kernel residente mínimo é carregado na memória. Em seguida, sempre que um usuário requisitar uma funcionalidade não está presente no kernel residente, um módulo do kernel, as vezes chamado de driver, é dinamicamente carregado na memória.

### **Exemplos De Módulos:**

Na imagem abaixo o **comando `lsmod`** Mostra o status atual de alguns módulos carregados no Kernel do Linux.

```
ac97_bus          12462      1 snd_ac97_codec
button            12817      0
ext4              302427      1
crc16             12327      1 ext4
jbd2              51626      1 ext4
mbcache           12938      1 ext4
usbhid            31704      0
hid               60188      1 usbhid
sg                21589      0
sr_mod            17468      0
cdrom             34813      1 sr_mod
sd_mod            35425      3
crc_t10dif        12332      1 sd_mod
ata_generic       12439      0
ata_piix          25392      0
ahci              24917      2
libahci           18494      1 ahci
ohci_hcd          22264      0
libata            125348      4 libahci,ahci,ata_piix,ata_generic
ehci_hcd          35702      0
usbcore           104793      4 ehci_hcd,ohci_hcd,usbhid
scsi_mod          135542      4 libata,sd_mod,sr_mod,sg
usb_common        12338      1 usbcore
e1000             76491      0
```

## Tutorial para o desenvolvimento de módulos para o kernel do Linux.

O tutorial criado para este trabalho ensina como criar um módulo para identificar quando um pen drive for inserido e mostrar algumas informações que podem vistas digitando **dmesg**.

O código fonte do módulo e o Makefile se encontram em anexo no repositório. Após adicionar ambos os arquivos citados acima basta usar o comando **sudo make** no diretório onde os mesmos se encontram.

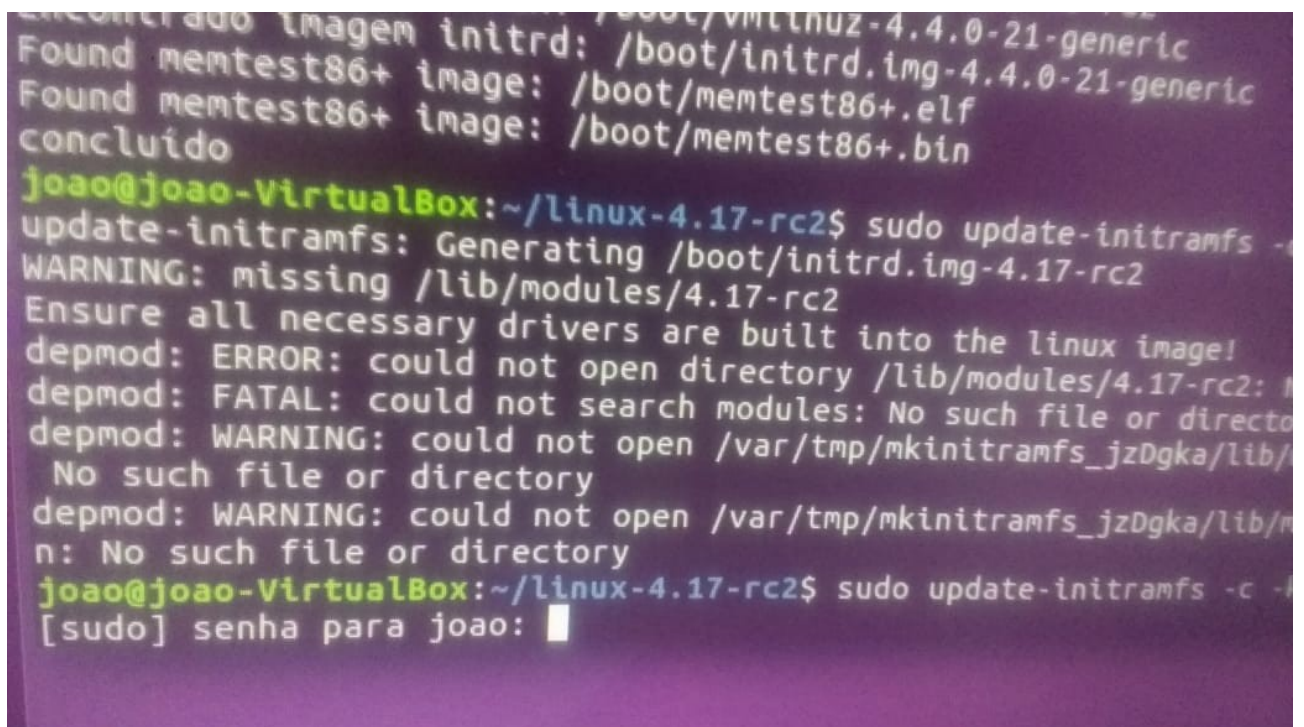
Caso o comando tenha sido executado corretamente será possível mais alguns arquivos no diretório usando o comando **ls**.

Para inserir o módulo é necessário utilizar o comando **sudo insmod (nome do arquivo).ko**

Para visualizar se o módulo carregado corretamente basta utilizar o comando **dmesg**. **OBS:** Para que o módulo funcione como esperado é preciso remover o módulo `usb_storage` utilizando o comando **sudo rmmod usb\_storage**.

## Modificar o kernel do Linux da distribuição (exemplo, Ubuntu 16.04 LTS) para suporte a sistemas de tempo real utilizando a aplicação RTAI1 (the RealTime Application Interface for Linux)

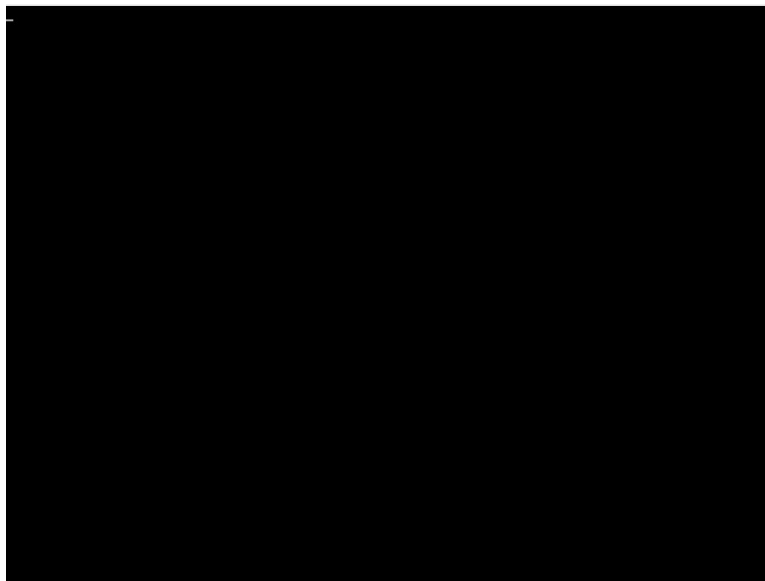
Apesar de terem sido feitas primeiro tentativas de compilar o Kernel do linux no ubuntu 16.04 cujo o kernel original é da versão 4.4.0-121 para que então se pudesse inserir o patch RTAI no kernel, não fomos capazes de compilar o kernel devido a alguns erros que ocorreram durante a compilação como mostrado abaixo.

A terminal window with a purple background. The prompt is 'joao@joao-VirtualBox:~/linux-4.17-rc2\$'. The user runs 'sudo update-initramfs -u'. The output shows 'update-initramfs: Generating /boot/initrd.img-4.17-rc2' followed by a 'WARNING: missing /lib/modules/4.17-rc2'. Then, 'depmod' errors are shown: 'ERROR: could not open directory /lib/modules/4.17-rc2:', 'FATAL: could not search modules: No such file or directory', and 'WARNING: could not open /var/tmp/mkinitramfs\_jzDgka/lib/No such file or directory'. The user then runs 'sudo update-initramfs -c' and is prompted for a password: '[sudo] senha para joao:'.

```
Found memtest86+ image: /boot/initrd.img-4.4.0-21-generic
Found memtest86+ image: /boot/memtest86+.elf
concluído
joao@joao-VirtualBox:~/linux-4.17-rc2$ sudo update-initramfs -u
update-initramfs: Generating /boot/initrd.img-4.17-rc2
WARNING: missing /lib/modules/4.17-rc2
Ensure all necessary drivers are built into the linux image!
depmod: ERROR: could not open directory /lib/modules/4.17-rc2: 
depmod: FATAL: could not search modules: No such file or directory
depmod: WARNING: could not open /var/tmp/mkinitramfs_jzDgka/lib/
No such file or directory
depmod: WARNING: could not open /var/tmp/mkinitramfs_jzDgka/lib/
n: No such file or directory
joao@joao-VirtualBox:~/linux-4.17-rc2$ sudo update-initramfs -c -f
[sudo] senha para joao: 
```

Na imagem acima nós tivemos problemas na hora de inserir o comando **sudo update-initramfs -c -k 4.17-rc2** seguindo o tutorial <https://www.linux.com/topic/desktop/how-compile-linux-kernel-0/> foram após o erro feito a tentativa de instalar os headers porém isso acabou não resolvendo o problema. **sudo apt-get install --reinstall linux-headers-\$(4.17)** os comandos **sudo apt-get update** && **sudo apt-get upgrade** já haviam sido inseridos antes.

Foi feita mais duas tentativas de compilar o kernel utilizando o tutorial <https://www.cyberciti.biz/tips/compiling-linux-kernel-26.html> onde foi feito a tentativa de compilar o kernel 5.9.14 porém após utilizarmos o comando **make** e aguardamos mais de 7 horas a máquina virtual acabou se corrompendo, a imagem abaixo mostra o que ocorreu logo após o sistema ser corrompido.



Mensagem exibida após tentarmos reiniciar a VM.

## Referencias

<https://www.certificacaolinux.com.br/como-funciona-o-kernel-do-linux/>

<https://www.linux.org/threads/understanding-the-linux-kernel.12508/>

<https://www.kernel.org/doc/html/latest/admin-guide/mm/index.html>