

PONTIFÍCIA UNIVERSIDADE CATÓLICA DE CAMPINAS

**CENTRO DE CIÊNCIAS EXATAS, AMBIENTAIS E DE
TECNOLOGIA**

CHRISTOPHER OLIVEIRA	RA:18726430
GIULIANO SANFINS	RA:17142837
MATHEUS MORETTI	RA:18082974
MURILO ARAUJO	RA:17747775
VICTOR REIS	RA:18726471

**SISTEMAS OPERACIONAIS B - EXPERIMENTO 1 -
COMPILAÇÃO DE KERNEL (LINUX)**

**CAMPINAS
2020**

SUMÁRIO

1. INTRODUÇÃO	3
2. DISCUSSÃO	3
2.1 Tempos de Execução:	9
2.2 Dificuldades:	9
3. CONCLUSÃO	10

1. INTRODUÇÃO

O experimento tem como objetivo principal a compilação de um Kernel Linux, permitindo através da mesma, que nós tenhamos um maior conhecimento e se adaptar a manipular e alterar recursos do Kernel, pois do decorrer do semestre vamos estar trabalhando e elaborando atividades mais complexas com o Kernel. Além de realizar a compilação, é esperado também com essa atividade que sejamos capazes de realizar a instalação e testes com um novo Kernel Linux.

E também estar entendendo todo o funcionamento de cada passo a passo do processo de compilação do kernel, conhecendo também, o tempo necessário para cada passo e os comandos e ferramentas utilizadas.

2. DISCUSSÃO

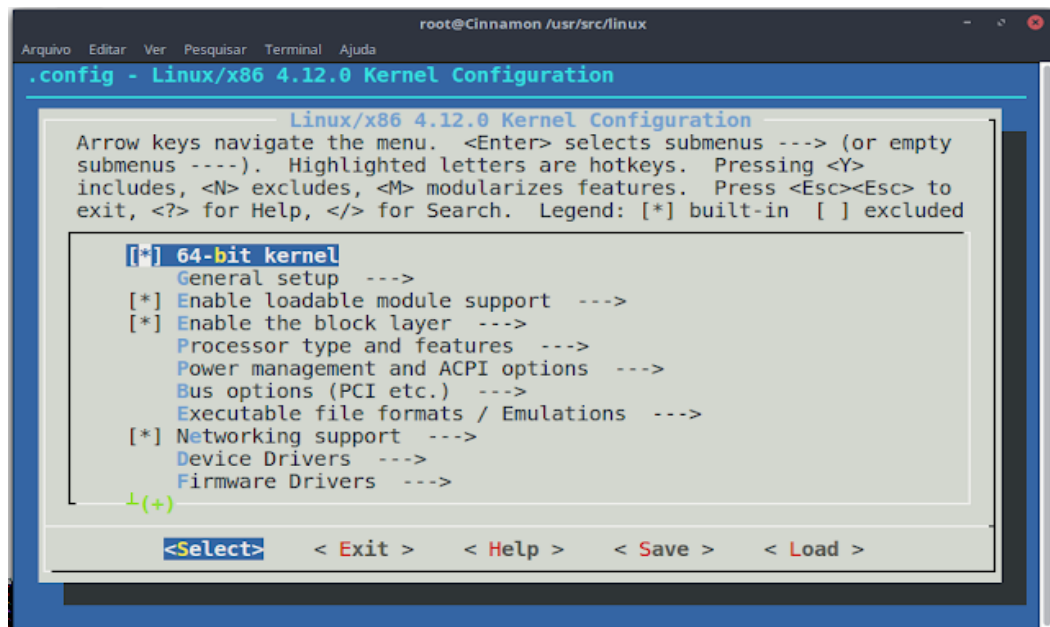
O primeiro passo foi a instalação de uma máquina virtual com o objetivo de simular um computador. O uso de máquinas virtuais traz como vantagem o menor uso de servidores e, com isso, uma economia em compra de equipamentos, em sua manutenção, em energia e em refrigeração do ambiente. Também reduz custos relacionados à instalação, atualização e manutenção de servidores. O segundo passo foi o download do kernel Ubuntu 4.1.15 usando os comandos:

```
apt-cache search linux-source  
apt-get versão
```

Depois da descompactação usamos o comando:

```
make menuconfig
```

O comando “*make menuconfig*” é uma ferramenta para que se possa compilar o código fonte do Linux. O lado positivo desse comando é que existe uma interface gráfica e intuitiva mostrando todas as configurações do Kernel(núcleo), como por exemplo, drivers de rede, drivers de vídeo, USB, WI-FI e etc. Exemplo da interface mostrada no *menuconfig*:



Um dos maiores objetivos desse experimento é modificar o Kernel para deixá-lo o mais enxuto possível para que a compilação seja rápida para o hardware desejado, sendo assim, necessário remover muitos drivers e configurações presentes no código fonte original do Linux. Essas configurações são salvas em um arquivo chamado .config” (Anexado com o relatório).

Depois de selecionar as configurações desejadas é necessário compilar o Kernel e deixar um arquivo no diretório *arch/x86_62/boot* chamado “*bzImage*”, que é o kernel comprimido.

```
make bzImage
```

```
root@khris-VirtualBox: /usr/src/linux-source-4.15.0
CHK scripts/mod/devicetable-offsets.h
CHK include/generated/timeconst.h
CHK include/generated/bounds.h
CHK include/generated/asm-offsets.h
CALL scripts/checksyscalls.sh
CHK include/generated/compile.h
Kernel: arch/x86/boot/bzImage is ready (#6)
root@khris-VirtualBox: /usr/src/linux-source-4.15.0# clear

root@khris-VirtualBox: /usr/src/linux-source-4.15.0# make bzImage -j4
CHK include/config/kernel.release
CHK include/generated/uapi/linux/version.h
warning: Cannot use CONFIG_STACK_VALIDATION=y, please install libelf-dev, libelf
-devel or elfutils-libelf-devel
CHK include/generated/utsrelease.h
CHK scripts/mod/devicetable-offsets.h
CHK include/generated/timeconst.h
CHK include/generated/bounds.h
CHK include/generated/asm-offsets.h
CALL scripts/checksyscalls.sh
CHK include/generated/compile.h
Kernel: arch/x86/boot/bzImage is ready (#6)
root@khris-VirtualBox: /usr/src/linux-source-4.15.0#
```

Após comprimir o kernel é necessário executar o comando “make modules_install” que garantirá a compilação dos módulos e instalar os módulos (binários) na pasta do kernel.

make modules_install

```
LD [M] net/sched/sch_qfq.ko
LD [M] net/sched/sch_red.ko
LD [M] net/sched/sch_sfb.ko
LD [M] net/sched/sch_sfq.ko
LD [M] net/sched/sch_tbf.ko
LD [M] net/sched/sch_teql.ko
LD [M] net/sctp/sctp.ko
LD [M] net/sctp/sctp_diag.ko
LD [M] net/smc/smc.ko
LD [M] net/sctp/sctp_probe.ko
LD [M] net/smc/smc_diag.ko
LD [M] net/sunrpc/auth_gss/auth_rpcgss.ko
LD [M] net/sunrpc/auth_gss/rpcsec_gss_krb5.ko
LD [M] net/sunrpc/sunrpc.ko
LD [M] net/sunrpc/xprtrdma/rpcrdma.ko
LD [M] net/tipc/tipc.ko
LD [M] net/tls/tls.ko
LD [M] net/unix/unix_diag.ko
LD [M] net/vmw_vsock/hv_sock.ko
LD [M] net/vmw_vsock/vmw_vsock_virtio_transport.ko
LD [M] net/vmw_vsock/vmw_vsock_virtio_transport_common.ko
LD [M] net/vmw_vsock/vsock.ko
LD [M] net/vmw_vsock/vsock_diag.ko
LD [M] net/x25/x25.ko
LD [M] net/xfrm/xfrm_algo.ko
LD [M] net/xfrm/xfrm_ipcomp.ko
LD [M] net/xfrm/xfrm_user.ko
LD [M] ubuntu/hio/hio.ko
LD [M] ubuntu/vbox/vboxguest/vboxguest.ko
LD [M] ubuntu/vbox/vboxsf/vboxsf.ko
LD [M] ubuntu/xr-usb-serial/xr_usb_serial_common.ko
LD [M] virt/lib/lrqbypass.ko
root@khris-VirtualBox: /usr/src/linux-source-4.15.0#
```

Agora que o Kernel já está compilado, os arquivos já podem ser copiados para seus destinos finais. O comando “*make install*” copiará o programa construído, *libraries* e documentação para seus destinos corretos.

```
make install
```

Depois que os módulos foram instalados precisamos “montar” o root *filesystem*, e para isso foi usado o comando *initramfs*.

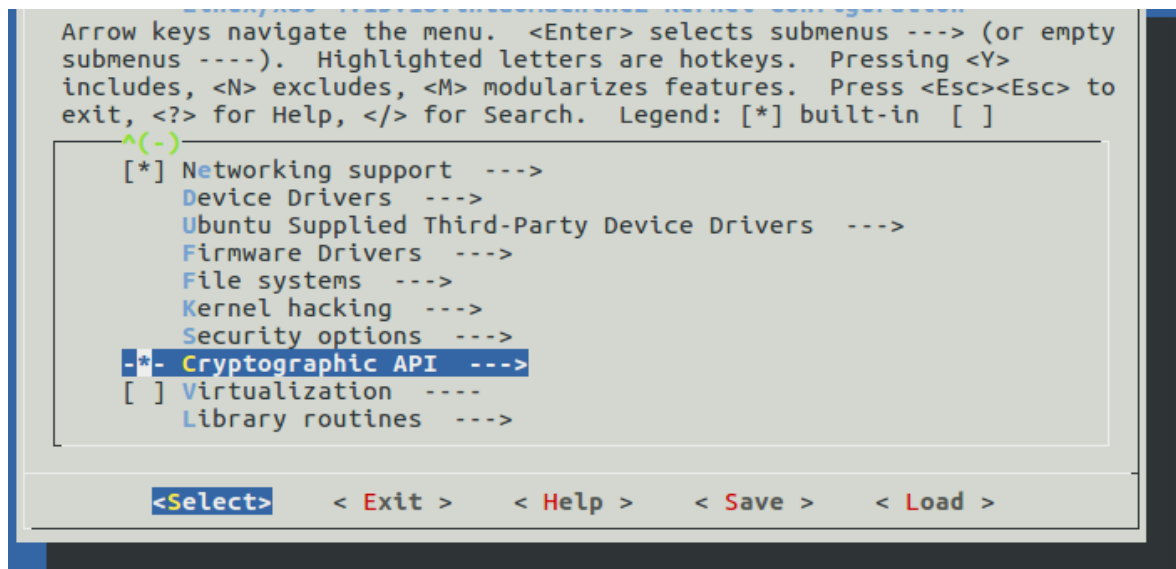
```
mkinitramfs -o initrd.img-4.12(use o tab para completar) (nome do kernel)
```

Com tudo instalado e montado só é preciso dar um upgrade no grub.

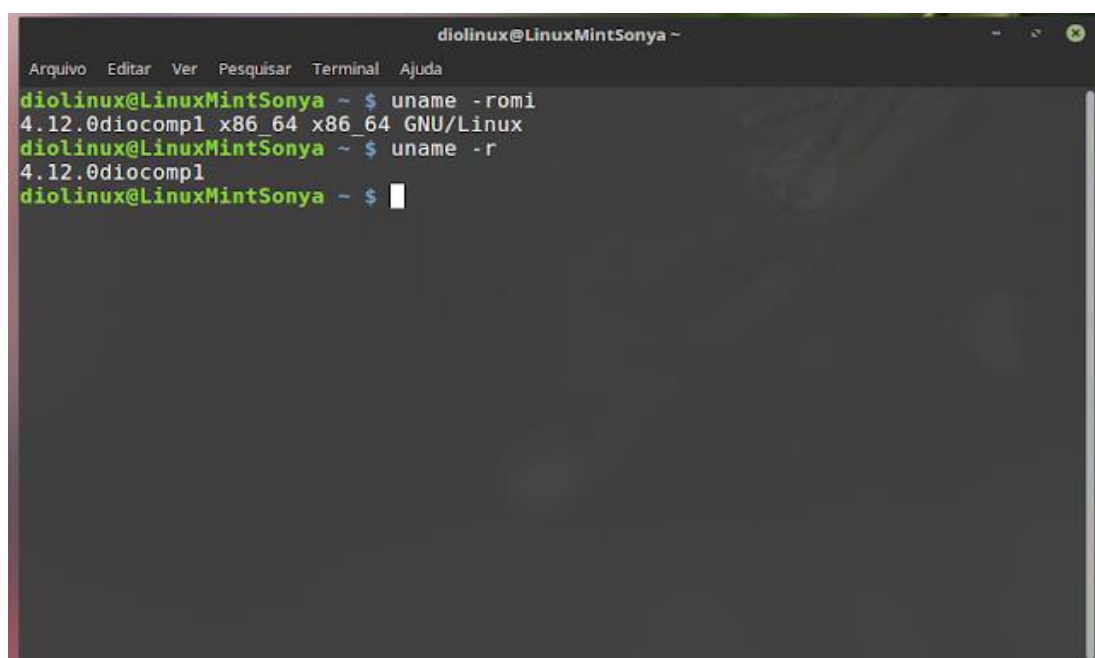
```
upgrade-grub
```

Feito tudo isso, já temos a imagem do kernel e do initrd encontradas com a nossa compilação nós já temos tudo pronto para começar a utilizar e testar o nosso kernel compilado. Após o reboot pode ser feito a verificação se o Kernel customizado foi realmente instalado com o comando:

```
uname -r
```



E o resultado deve ser algo parecido (se você mudou o nome da versão que fica armazenado no arquivo Makefile);



Interface do MenuConfig.

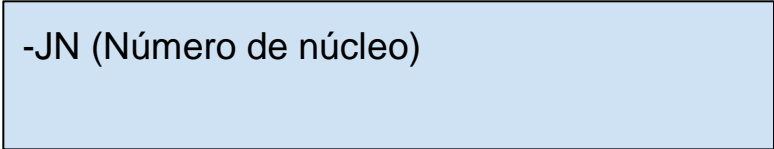
Retorno após criação da Imagem.

```
root@khrris-VirtualBox: /usr/src/linux-source-4.15.0
CHK      include/generated/compile.h
UPD      include/generated/compile.h
CC       init/version.o
AR       init/built-in.o
AR       built-in.o
LD       vmlinux.o
MODPOST  vmlinux.o
KSYM     .tmp_kallsyms1.o
KSYM     .tmp_kallsyms2.o
LD       vmlinux
SORTEX   vmlinux
SYSMAP   System.map
CC       arch/x86/boot/version.o
VOFFSET  arch/x86/boot/compressed/./voffset.h
OBJCOPY  arch/x86/boot/compressed/vmlinux.bin
RELOCS   arch/x86/boot/compressed/vmlinux.relocs
CC       arch/x86/boot/compressed/kaslr.o
CC       arch/x86/boot/compressed/misc.o
GZIP     arch/x86/boot/compressed/vmlinux.bin.gz
MKPIGGY  arch/x86/boot/compressed/piggy.S
AS       arch/x86/boot/compressed/piggy.o
LD       arch/x86/boot/compressed/vmlinux
ZOFFSET  arch/x86/boot/zoffset.h
OBJCOPY  arch/x86/boot/vmlinux.bin
AS       arch/x86/boot/header.o
LD       arch/x86/boot/setup.elf
OBJCOPY  arch/x86/boot/setup.bin
BUILD    arch/x86/boot/bzImage
Setup is 17212 bytes (padded to 17408 bytes).
System is 7778 kB
CRC bb642908
Kernel: arch/x86/boot/bzImage is ready (#7)
root@khrris-VirtualBox: /usr/src/linux-source-4.15.0#
```

Retorno do Comando Make Modules.

```
Terminal File Edit View Search Terminal Help
LD [M] net/sched/sch_qfq.ko
LD [M] net/sched/sch_red.ko
LD [M] net/sched/sch_sfb.ko
LD [M] net/sched/sch_sfq.ko
LD [M] net/sched/sch_tbf.ko
LD [M] net/sched/sch_teql.ko
LD [M] net/sctp/sctp.ko
LD [M] net/sctp/sctp_diag.ko
LD [M] net/smc/smc.ko
LD [M] net/sctp/sctp_probe.ko
LD [M] net/smc/smc_diag.ko
LD [M] net/sunrpc/auth_gss/auth_rpcgss.ko
LD [M] net/sunrpc/auth_gss/rpcsec_gss_krb5.ko
LD [M] net/sunrpc/sunrpc.ko
LD [M] net/sunrpc/xprtrdma/rpcrdma.ko
LD [M] net/tipc/tipc.ko
LD [M] net/tls/tls.ko
LD [M] net/unix/unix_diag.ko
LD [M] net/vmw_vsock/hv_sock.ko
LD [M] net/vmw_vsock/vmw_vsock_virtio_transport.ko
LD [M] net/vmw_vsock/vmw_vsock_virtio_transport_common.ko
LD [M] net/vmw_vsock/vsock.ko
LD [M] net/vmw_vsock/vsock_diag.ko
LD [M] net/x25/x25.ko
LD [M] net/xfrm/xfrm_algo.ko
LD [M] net/xfrm/xfrm_ipcomp.ko
LD [M] net/xfrm/xfrm_user.ko
LD [M] ubuntu/hio/hio.ko
LD [M] ubuntu/vbox/vboxguest/vboxguest.ko
LD [M] ubuntu/vbox/vboxsf/vboxsf.ko
LD [M] ubuntu/xr-usb-serial/xr_usb_serial_common.ko
LD [M] virt/lib/irqbypass.ko
root@khrris-VirtualBox: /usr/src/linux-source-4.15.0#
```


Também foi realizado *overclock* na máquina através das configurações da *bios* e utilizado o comando abaixo para escolher 4 (quatro) núcleos para a realizar a compilação do kernel da máquina virtual.



-JN (Número de núcleo)

2.1 Tempos de Execução:

Computador 1.

Config:

Processador: i5-7600k (4.9Ghz) (Com OC)

Memória RAM: 20Gb.

Execuções e seus tempos:

Imagem do Kernel: 6:20 min.

Compilação do Kernel: 12:25 min.

Tempo total: 18:45 min.

2.2 Dificuldades:

- Superaquecimento do processador, chegando até 95°C.
- Processamento do computador.

3. CONCLUSÃO

Com esse experimento pudemos conhecer mais sobre o *linux* no que se diz respeito ao kernel, como procurar kernels disponíveis, baixá-lo e instalá-lo, do mesmo modo aprendemos a como verificar e alterar as configurações presentes no *kernel*, e através disso pudemos concluir um dos principais objetivos do experimento, que foi deixar todo processo de compilação de modules, e instalação do kernel muito mais rápido. Assim de forma geral inferimos que fora alcançado todos os objetivos propostos para esse experimento, tais como, aprender mais sobre o sistema *linux* voltado a âmbito de kernel, aprender os processos envolvidos desde a busca de um kernel até sua configuração e instalação para uso, e a como distinguir quais opções presentes nas configurações são realmente necessárias para que possamos cumprir nosso futuros objetivos de forma que exigirá menos tempo.