

**Sistemas Operacionais B**

**Professor: Prof. Edmar Rezende**

**Experimento 1 - Instalação do Kernel**

**Alunos:**      **Gabriel Gonçalves Mattos Santini - 18189084**  
                 **Luiz Vinicius dos Santos Ruoso - 18233486**  
                 **Marcelo Germani Olmos - 18048298**  
                 **Victor Felipe dos Santos - 18117820**  
                 **Victor Luiz Fraga Soldera - 18045674**

**Introdução:**

Foi proposto pelo professor, a realização da compilação do Kernel do Linux com o principal objetivo de aprender o que é necessário para realizar a compilação do mesmo, experienciar suas falhas e como contorná-las. Dessa forma ganhando experiência nessa tarefa. Para a realização da tarefa, foi usado o Sistema Operacional Ubuntu derivado no *Linux*, na Versão 16.04 desse SO

## Desenvolvimento

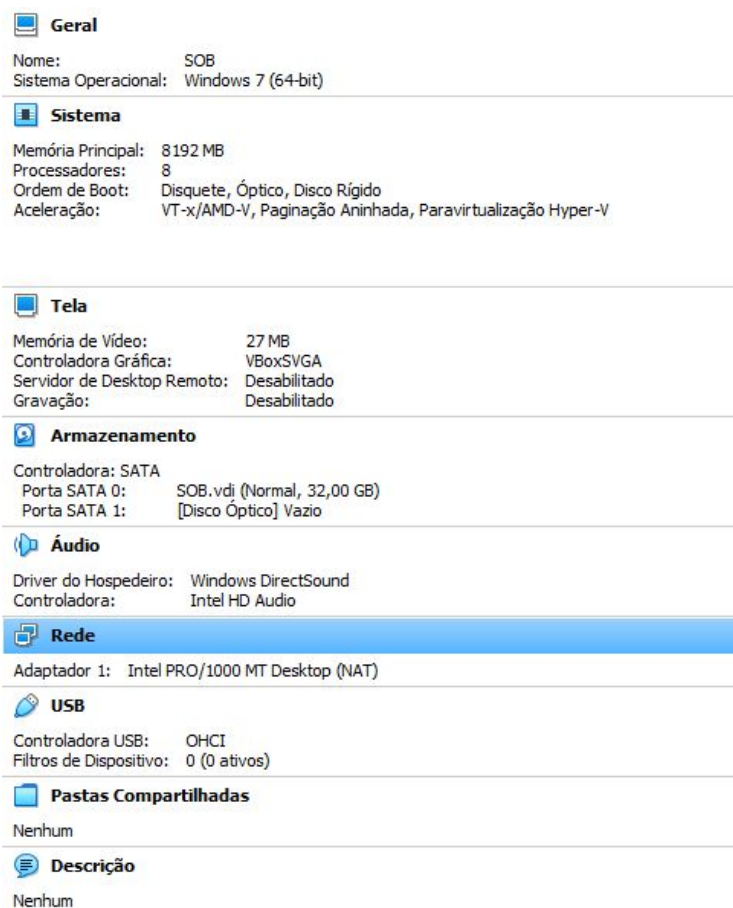
### Instalação e configuração da máquina virtual:

Para o experimento em questão utilizamos a máquina virtual Virtual Box (disponível em: <https://www.virtualbox.org/>). Para o desenvolvimento do experimento, precisamos fazer o download do sistema operacional Ubuntu, em sua versão 16.04 (disponível em: <https://releases.ubuntu.com/16.04/>) e instalá-lo na máquina virtual, utilizando de um .ISO.

Após a instalação, dentro do programa da máquina virtual, nomeou-se a máquina virtual de SOB:



Nas configurações do integrante Victor Soldera, por exemplo, configuramos da seguinte forma:



## Passos para compilação e execução do Kernel:

Primeiramente, foi necessário o download do pacote (package) *libncurses* utilizando o comando de terminal (*Control + T*) `sudo apt-get install libncurses5-dev`.

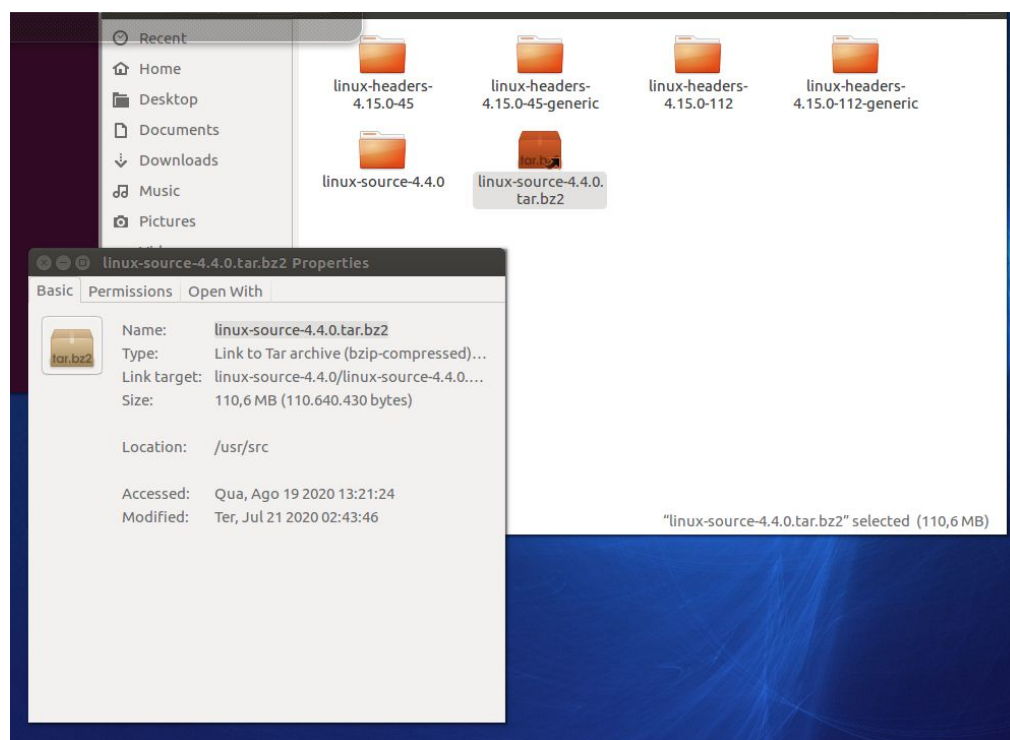
```
raven@raven-VirtualBox:~$ sudo apt-get install libncurses5-dev
Reading package lists... Done
Building dependency tree
Reading state information... Done
libncurses5-dev is already the newest version (6.0+20160213-1ubuntu1).
0 upgraded, 0 newly installed, 0 to remove and 73 not upgraded.
```

Após este comando, fizemos o download do Kernel ([versão 4.4.0](#)) utilizando o comando `sudo apt-get install linux-source-4.4.0`.

```
raven@raven-VirtualBox:~$ sudo apt-get install linux-source-4.4.0
Reading package lists... Done
Building dependency tree
Reading state information... Done
linux-source-4.4.0 is already the newest version (4.4.0-187.217).
0 upgraded, 0 newly installed, 0 to remove and 73 not upgraded.
```

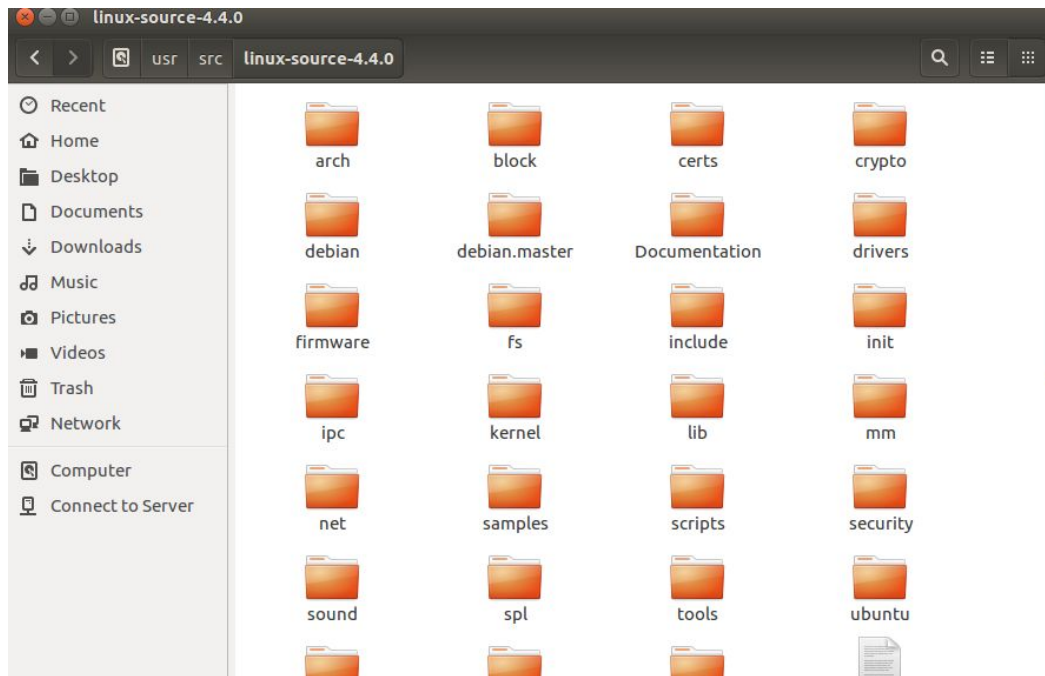
Em seguida, seguimos para o seguinte caminho: `/usr/src/`

Dentro do caminho havia o `linux-source-4.4.0.tar.bz2`

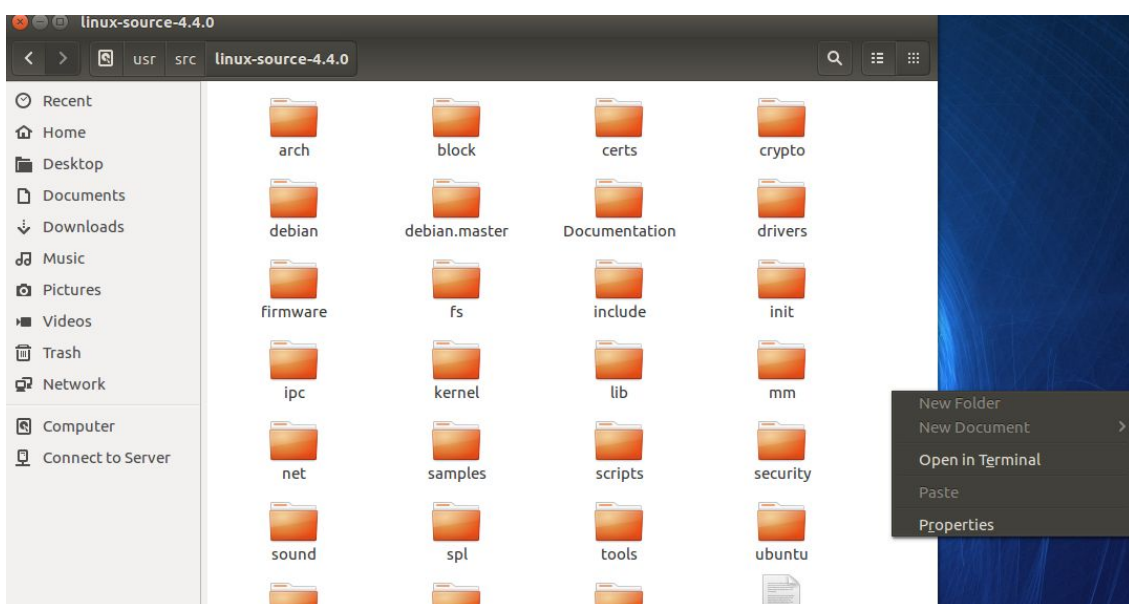


Para acessar os arquivos contidos no `.tar.bz2` utilizamos `tar xjpf linux-4.4.0.tar.bz2` que descompacta o mesmo e cria uma pasta de mesmo nome: `linux-source-4.4.0`

Entramos dentro da pasta `linux-source-4.4.0`



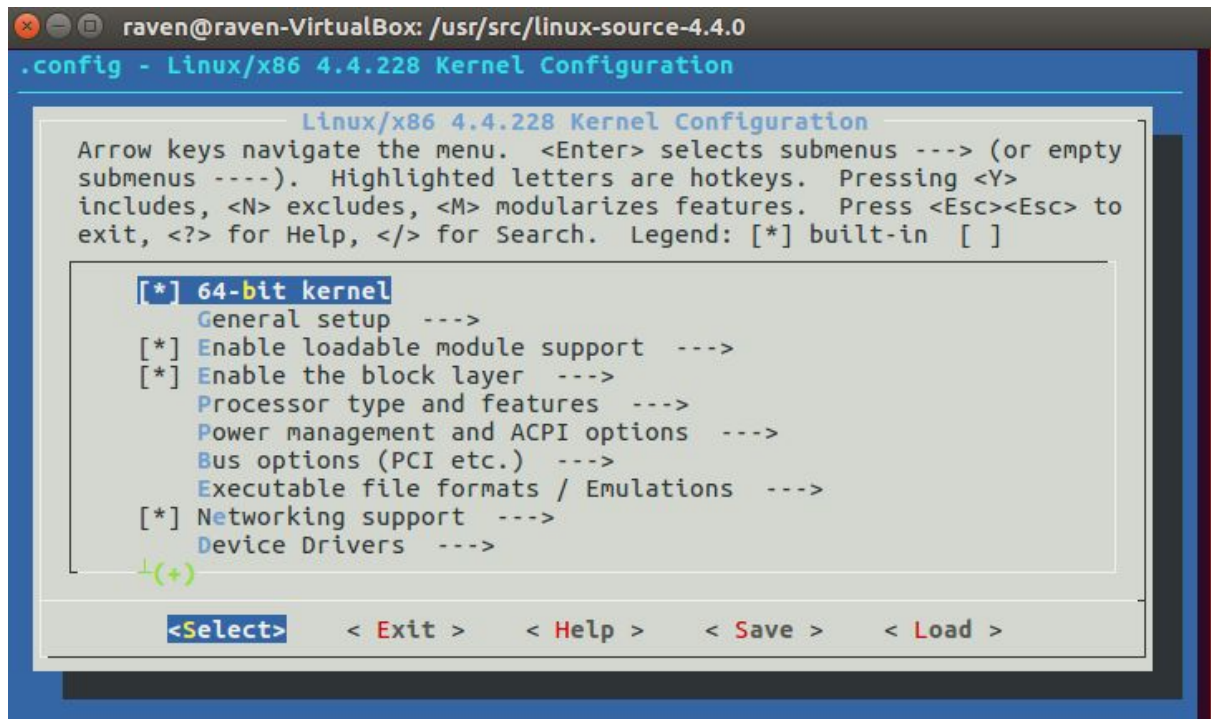
Dentro da mesma, abrimos o terminal utilizando o botão direito:



Clicamos em: *Open in Terminal*

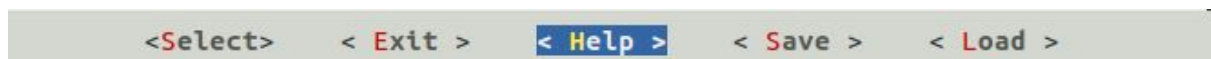
Dentro do terminal, comandamos *sudo make menuconfig*

Assim, abrindo as configurações do Kernel, que usaremos para retirar módulos do mesmo:



Para diminuir o tempo de compilação do kernel, desativamos vários módulos que julgamos desnecessários para o funcionamento desejado do sistema, com a ajuda da opção help do menuconfig, como na imagem abaixo:

Alguns dos módulos, em help, davam-nos dicas se deveríamos ou não desativá-los. Abaixo segue imagem do help dizendo que: “Se não tiver certeza, habilite-o”:





```
raven@raven-VirtualBox: /usr/src/linux-source-4.4.0
.config - Linux/x86 4.4.228 Kernel Configuration
> Device Drivers > Device Tree and Open Firmware support
Device Tree and Open Firmware support

CONFIG_OF:

This option enables the device tree infrastructure.
It is automatically selected by platforms that need it or can
be enabled manually for unittests, overlays or
compile-coverage.

Symbol: OF [=n]
Type : boolean
Prompt: Device Tree and Open Firmware support
Location:
-> Device Drivers
Defined at drivers/of/Kconfig:4
Selected by: X86_INTEL_CE [=n] && PCI [=y] && PCI_GODIRECT [=n] && \
X86_IO_APIC [=y] && X86_32 [=n] && X86_EXTENDED_PLATFORM [=y] || \
OLPC [=n] && X86_32 [=n] && !X86_PAE [=n]

( 99%)
< Exit >
```

Alguns módulos, nos diziam: “Se não tiver certeza, diga sim”

```
raven@raven-VirtualBox: /usr/src/linux-source-4.4.0
.config - Linux/x86 4.4.228 Kernel Configuration
> Device Drivers
Plug and Play support

CONFIG_PNP:

Plug and Play (PnP) is a standard for peripherals which allows those
peripherals to be configured by software, e.g. assign IRQ's or other
parameters. No jumpers on the cards are needed, instead the values
are provided to the cards from the BIOS, from the operating system,
or using a user-space utility.

Say Y here if you would like Linux to configure your Plug and Play
devices. You should then also say Y to all of the protocols below.
Alternatively, you can say N here and configure your PnP devices
using user space utilities such as the isapnptools package.

If unsure, say Y.

Symbol: PNP [=y]

( 69%)
< Exit >
```

Outros nos diziam: “Se não tiver certeza, não habilite-o”:

```
raven@raven-VirtualBox: /usr/src/linux-source-4.4.0
.config - Linux/x86 4.4.228 Kernel Configuration
> Device Drivers

Memory Technology Device (MTD) support

CONFIG_MTD:

Memory Technology Devices are flash, RAM and similar chips, often
used for solid state file systems on embedded devices. This option
will provide the generic support for MTD drivers to register
themselves with the kernel and for potential users of MTD devices
to enumerate the devices which are present and obtain a handle on
them. It will also allow you to select individual drivers for
particular hardware and users of MTD devices. If unsure, say N.

Symbol: MTD [=m]
Type : tristate
Prompt: Memory Technology Device (MTD) support
Location:
-> Device Drivers
Defined at drivers/mtd/Kconfig:1

( 95%)
< Exit >
```

E outros nos diziam: “Não use esses testes se você não souber exatamente o que fazer”:

```
raven@raven-VirtualBox: /usr/src/linux-source-4.4.0
.config - Linux/x86 4.4.228 Kernel Configuration
> Device Drivers > Memory Technology Device (MTD) support

MTD tests support (DANGEROUS)

CONFIG_MTD_TESTS:

This option includes various MTD tests into compilation. The tests
should normally be compiled as kernel modules. The modules perform
various checks and verifications when loaded.

WARNING: some of the tests will ERASE entire MTD device which they
test. Do not use these tests unless you really know what you do.

Symbol: MTD_TESTS [=n]
Type : tristate
Prompt: MTD tests support (DANGEROUS)
Location:
-> Device Drivers
-> Memory Technology Device (MTD) support (MTD [=m])
Defined at drivers/mtd/Kconfig:15

( 92%)
< Exit >
```

Para remover os módulos, utilizamos esta [tabela](#), para diferenciar todos os símbolos presentes dentro do `menuconfig`:

## Usage

In the shown menu the blue bar indicates the position of the cursor. With the **↑** and **↓** arrow keys change the position of the cursor. The **←** and **→** arrow keys traverse the menu bar in the bottom and define what happens when the **Enter** key is pressed. For the menu bar below, **Select** switches to a sub menu for the menu entries ending with **---** while **Exit** exits a sub menu. As an alternative the **Esc** key can be pressed twice to exit the application.

Pressing an associated letter key **A-Z** will move the position of the cursor lines that have characters in bold. The **Y**, **M**, **N** keys are excluded from navigation in this way; they are sanctified for other purposes. If a line begins with a Y, M, or N, the next character will be bold and capable of being jumped to. For example, relative to the cursor's current position, if the next line reads "Network Device Support --->" pressing the **E** key will move the cursor to that line.

The following symbols can appear in front of the lines in the menus:

Symbol(s)	Description
[ ], [*]	Options in square brackets can be activated or deactivated. The asterisk marks the menu entry as activated. The value can be changed with the <b>space</b> key. It is also possible to press <b>Y</b> key (Yes) to activate or <b>N</b> key (No) to deactivate the selected entry.  If the option is activated, the selected feature/driver will be built into the kernel and will always be available at boot time.
< >, <M>, <*>	Options in angle brackets can be activated or deactivated, but also activated as module (indicated by a <i>M</i> ). The values can be modified by pressing <b>Y</b> / <b>N</b> keys as before or by pressing the <b>M</b> key to activate the feature/driver as a module.  See the <a href="#">Kernel Modules</a> article for differentiation.
{M}, {*}	Options in curly brackets can be activated or activated as module but not be deactivated. This happens because another feature/driver is dependent on this feature.
-M-, -*-	Options between hyphens are activated in the shown way by another feature/driver. There is no choice.

Furthermore some menu entries have a tag at the end:

Tag	Description
(NEW)	This driver is new in the kernel and is maybe not stable enough.
(EXPERIMENTAL)	This driver is experimental and most likely not stable enough.
(DEPRECATED)	This driver is deprecated and not needed for most systems.
(OBSOLETE)	This driver is obsolete and should not be activated.

Most options have a description, which see by pressing the **H** key or choosing **Help** in the menu bar.

Para conseguirmos compilar o kernel configurado, necessitamos de instalar o seguinte pacote: `sudo apt-get install libssl-dev`.

Feita a instalação do pacote, utilizamos o comando `sudo make`, de forma a compilar os módulos do kernel escolhidos

Para compilar o kernel utilizamos o comando `make -jN`, com **N** sendo o número de cores que o usuário deseja utilizar ao realizar a compilação. Assim, em nosso caso, o tempo para compilar o kernel foi reduzido em aproximadamente 50%; No exemplo abaixo, usaram-se 8 cores para a compilação:



```
CC [M] drivers/gpu/drm/i915/i915_gem_evict.o
CC [M] net/ipv4/netfilter/nft_masq_ipv4.o
CC drivers/mailbox/pcc.o
CC [M] drivers/infiniband/hw/ocrdma/ocrdma_ah.o
CC net/ipv6/raw.o
CC [M] drivers/mailbox/mailbox-altera.o
CC [M] drivers/gpu/drm/i915/i915_gem_execbuffer.o
CC [M] net/ipv4/netfilter/nft_redir_ipv4.o
LD drivers/mailbox/built-in.o
LD net/ipx/built-in.o
CC [M] drivers/infiniband/hw/ocrdma/ocrdma_stats.o
CC [M] net/ipx/af_ipx.o
CC [M] fs/nfs/nfs4state.o
CC [M] net/ipv4/netfilter/nft_dup_ipv4.o
CC net/ipv6/icmp.o
CC [M] fs/nfs/nfs4renewd.o
LD [M] drivers/infiniband/hw/ocrdma/ocrdma.o
LD drivers/infiniband/hw/qib/built-in.o
CC [M] drivers/infiniband/hw/qib/qib_cq.o
LD drivers/infiniband/hw/usnic/built-in.o
CC [M] drivers/infiniband/hw/usnic/usnic_fwd.o
CC [M] net/ipv4/netfilter/nf_tables_arp.o
CC [M] drivers/gpu/drm/i915/i915_gem_fence.o
```

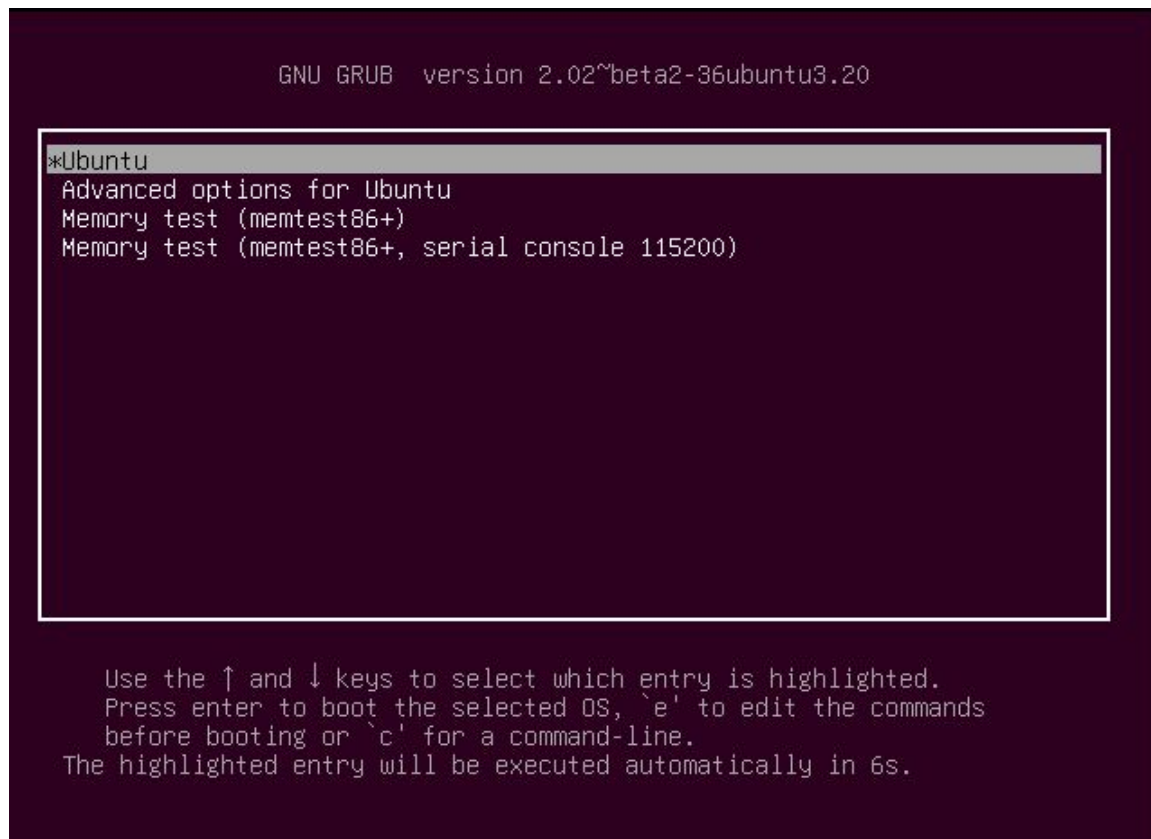
Em seguida, utilizamos o comando: *sudo make modules\_install* para ter certeza se os binários foram realmente criados (se não, criá-los) e em seguida os instalar no diretório do kernel compilado.

Com isso, usamos o comando *sudo make install*, que instala os arquivos necessários no diretório */boot*, assim como faz as modificações essenciais nas configurações do *Grub*.

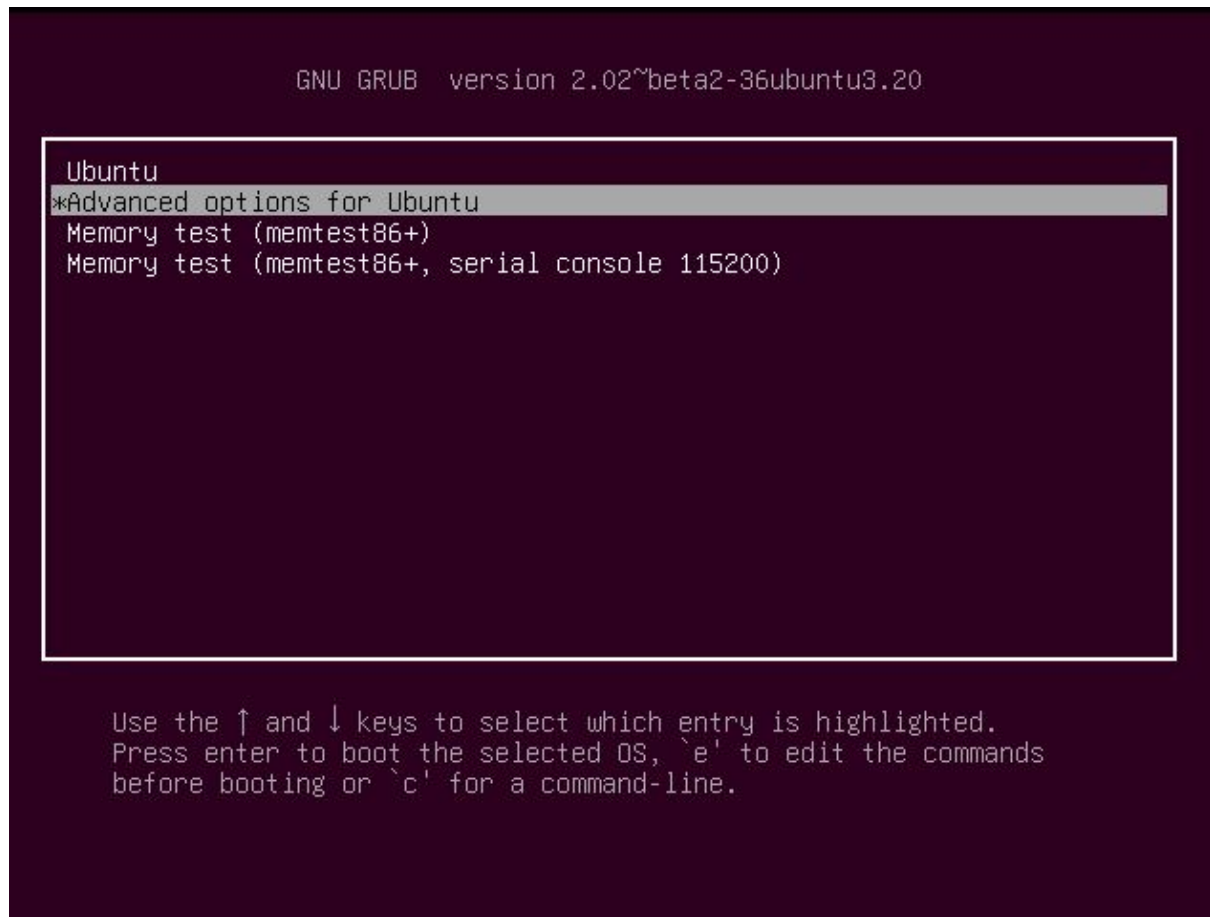
Para finalizar, utilizamos o comando *sudo update-grub*, para que o grub fosse atualizado, permitindo, assim, que pudéssemos abrir o kernel pelo mesmo.

## Teste do Kernel:

Para testar o kernel, após a compilação e instalação do mesmo, reiniciamos a máquina virtual, para que, assim, o grub iniciasse:



Selecionamos a opção: *Advanced options for Ubuntu* utilizando as setas do teclado.



Dentro das configurações avançadas, vimos que nosso kernel havia sido compilado, no exemplo, com o nome de *Linux-4.4.228SOB1.1-removed-some-network-configs*

```
GNU GRUB  version 2.02~beta2-36ubuntu3.20

Ubuntu, with Linux 4.15.0-112-generic
Ubuntu, with Linux 4.15.0-112-generic (upstart)
Ubuntu, with Linux 4.15.0-112-generic (recovery mode)
Ubuntu, with Linux 4.15.0-45-generic
Ubuntu, with Linux 4.15.0-45-generic (upstart)
Ubuntu, with Linux 4.15.0-45-generic (recovery mode)
*Ubuntu, with Linux 4.4.228SOB1.1-removed-some-network-configs
Ubuntu, with Linux 4.4.228SOB1.1-removed-some-network-configs (upstart)
Ubuntu, with Linux 4.4.228SOB1.1-removed-some-network-configs (recovery)
Ubuntu, with Linux 4.4.228SOB1.1-removed-some-network-configs.old
Ubuntu, with Linux 4.4.228SOB1.1-removed-some-network-configs.old (upst
Ubuntu, with Linux 4.4.228SOB1.1-removed-some-network-configs.old (reco

Use the ↑ and ↓ keys to select which entry is highlighted.
Press enter to boot the selected OS, `e' to edit the commands
before booting or `c' for a command-line. ESC to return previous
menu.
```

Assim, pressionando *enter*, o sistema iniciará o kernel que instalamos.

## Tempos obtidos por Hardware:

### Até o *sudo update-grub*

#### Luiz Vinícius:

#### Configurações:

CPU Intel(R) Core(TM) i5-4690 CPU @ 3.50GHz Velocidade base: 3,50 GHz  
Sockets: 1 Núcleos: 4 Processadores lógicos: 4. Cache L1: 256 KB Cache  
L2: 1,0 MB Cache L3: 6,0 MB

Memória 16 GB DDR3. Velocidade: 1333 MHz.

Memória Secundária: 999 GB HD.



#### Configurações Virtualizadas:

CPU: 3.50 GHz com 4 núcleos;  
RAM: 6 GB;  
Memória Secundária: 40 GB HD.

Tempo obtido: 40min

#### Victor Soldera:

#### Configurações:

CPU: AMD Ryzen 5 2600X 3.6 GHz, com 6 cores, 12 threads e 12 processadores lógicos: Cache L1: 576KB, Cache L2: 3.0MB, Cache L3: 16MB

Memória: 16GB DDR4 Dual Channel. Velocidade: 2666 MHz

Memória Secundária: 1TB HD

#### Configurações Virtualizadas:

CPU: 3.6GHz com 8 núcleos;  
RAM: 8GB  
Memória Secundária: 32GB HD

Tempo obtido: 12min.

#### Victor Felipe:

#### Configurações:

CPU: Intel(R) Core(TM) i5-8600K CPU @ 3.60GHz, com 6 cores Velocidade base: 3,60 GHz Sockets: 1 Núcleos: 6 Processadores lógicos: 6. Cache L1: 384 KB Cache L2: 1,5 MB Cache L3: 9,0 MB

Memória 16GB DDR4. Velocidade: 2133 MHz.

Memória Secundária: 1TB HD.

#### Configurações Virtualizadas:

CPU: 3.50 GHz com 6 núcleos;  
RAM: 6 GB;  
Memória Secundária: 40 GB HD.

Tempo obtido: 12min

Gabriel Santini:

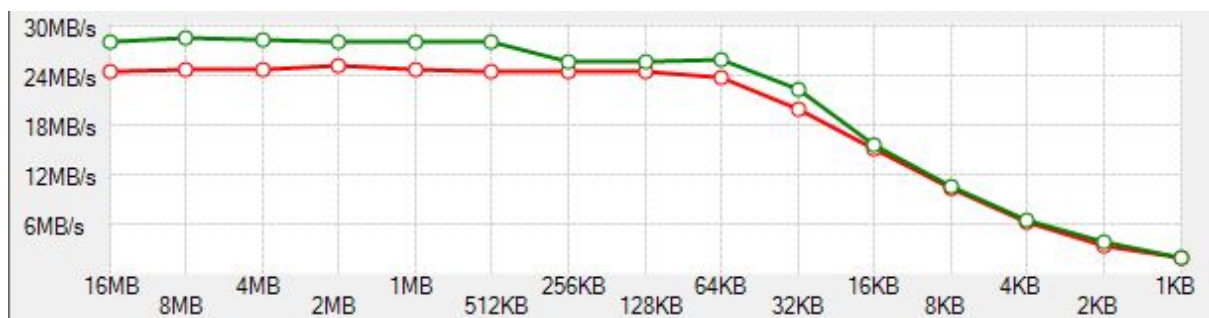
Configurações :

CPU Intel(R) Core(TM) i7-3770K CPU @ 3.50GHz Velocidade base: 3,50  
GHz Sockets: 1 Núcleos: 4 Processadores lógicos: 8 Cache L1: 256 KB  
Cache L2: 1,0 MB Cache L3: 8,0 MB.  
Memória 8,0 GB DDR3 Velocidade: 1333 MHz.

Configurações virtualizadas:

CPU:3,5GHz com 4 núcleos  
RAM:4GB  
Memória Secundária: 50GB HD  
Tempo obtido:1h10min

Gabriel utilizou um hd externo de, em média 25 MB/s (Megabytes por segundo), ou seja, a demora foi devido ao gargalo entre o HD Externo e o processador(Intel(R) Core(TM) i7-3770K CPU @ 3.50GHz). Abaixo imagens do teste de benchmark do HD externo e do HD:



Velocidade de escrita e leitura HD Externo por tamanho do arquivo



Velocidade de escrita de leitura do HD pelo tamanho do arquivo

### Dificuldades Encontradas

Usando o software VMWare (gerador de instâncias para máquinas virtuais), encontramos problemas na inicialização do kernel compilado, sendo esse kernel um preparado por nós tendo sido retirado alguns módulos de Drivers. O erro abaixo foi retirado de [‘Sobre Linux’](#) e representa o erro que experienciamos:

Gave up waiting for root device. Common problems:

- Boot args (cat /proc/cmdline)
- Check rootdelay= (did the system wait long enough?)
- Check root= (did the system wait for the right device?)
- Missing modules (cat /proc/modules; ls /dev)

ALERT! /dev/disk/by-uuid/aa91b9fe-1e27-44d7-9c1b-72dd7d4e8575 does not exist.  
Dropping to a shell!

BusyBox v.1.13.3 (Ubuntu 1:1.13.3-1ubuntu11) built-in shell (ash)  
Enter 'help' for list of built-in commands.

(initramfs)

Conseguimos corrigir este erro ao desativar a verificação de *UUID* (identificação única para a partição do disco) pelo *Grub*, em */etc/default/grub*, retira-se o comentário (#) de *GRUB\_DISABLE\_LINUX\_UUID = true*;

Após isso, criou-se outro erro, onde dizia-se **Alert! /dev/sda1 does not exist.**

Este erro foi um impedimento para continuar usando a VMWare, já que não conseguimos encontrar soluções o mesmo. Então, mudamos para a Oracle VirtualBox onde não houve estes problemas.

Além disso, no próprio *Kernel* (4.4.0) há uma *bug* já conhecido, que retornaria o seguinte erro: error: 'ovl\_v1\_fs\_type' undeclared


Para contornar este possível erro, entramos *.config* em modo administrador (*sudo*) e habilitamos `CONFIG_OVERLAY_FS_V1`, igualando-a a um, como em: `CONFIG_OVERLAY_FS_V1=1;`

Durante a compilação do kernel, usando o comando *sudo make*, obtivemos um erro:

```
$ cp ./cp210x.ko /lib/modules/4.8.0-58-generic/kernel/drivers/usb/serial/cp210x.ko $
insmod /lib/modules/4.8.0-58-generic/kernel/drivers/usb/serial/cp210x.ko insmod:
ERROR: could not insert module
/lib/modules/4.8.0-58-generic/kernel/drivers/usb/serial/cp210x.ko: Unknown symbol
in module
```

Isto foi causado por desabilitamos o *USB Support* dentro do *menuconfig*; Assim, quando o ativamos novamente, o erro foi corrigido.

Durante a compilação do kernel (*sudo make*) houve o seguinte erro: *openssl/opensslv.h: No such file or directory*.

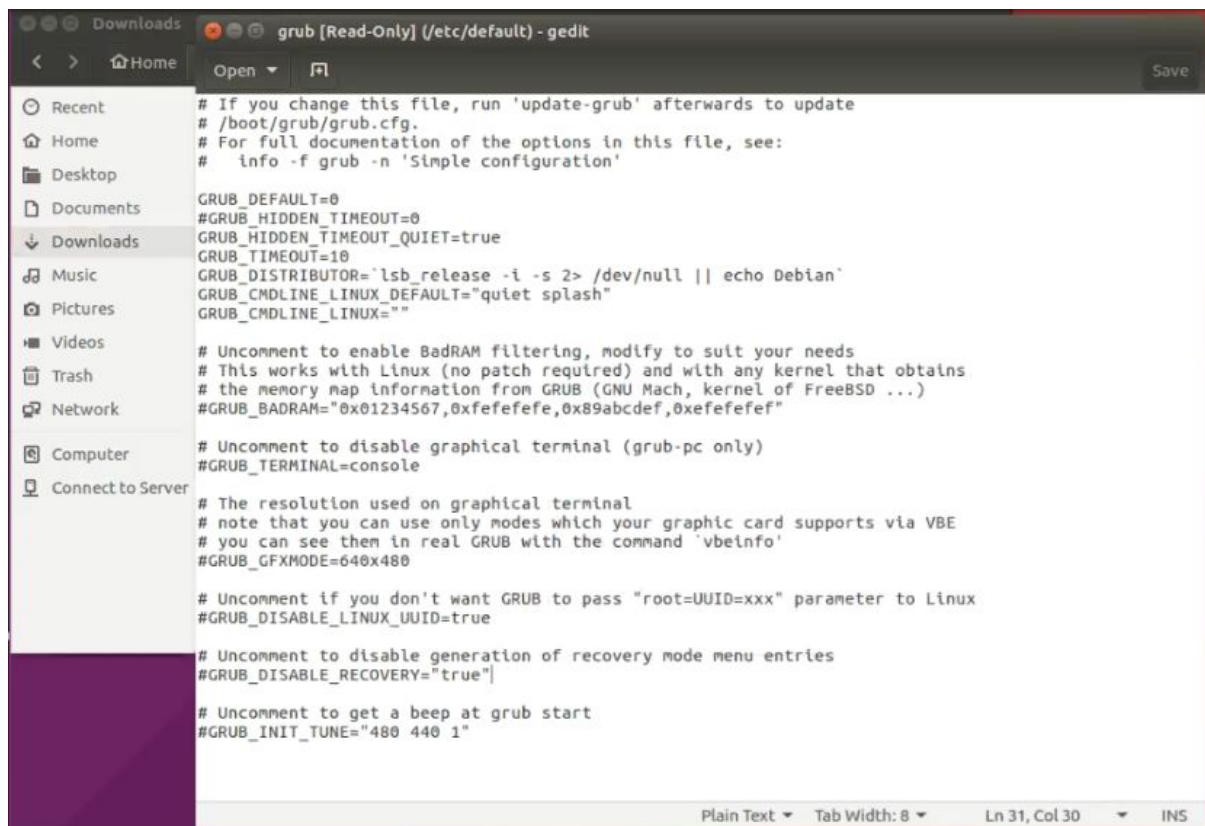


```
scripts/sign-file.c:23:30: fatal error: openssl/opensslv.h: No such file or directory
compilation terminated.
scripts/Makefile.host:91: recipe for target 'scripts/sign-file' failed
make[1]: *** [scripts/sign-file] Error 1
Makefile:566: recipe for target 'scripts' failed
make: *** [scripts] Error 2
```

Pesquisando pelo mesmo, havia-se de se instalar a biblioteca *libssl-dev*, utilizando o seguinte comando: *sudo apt-get install libssl-dev*

Outro problema encontrado, foi com não inicialização do grub. Para corrigi-lo, acessamos o caminho */etc/default/* e acessamos o grub no gedit utilizando o comando *sudo gedit grub*, permitindo a escrita no arquivo do grub. Com isso feito, alteramos o valor de *GRUB\_DEFAULT* para 0 ou -1, adicionamos um # em *GRUB\_HIDDEN\_TIMEOUT* e deixamos *GRUB\_CMDLINE\_LINUX=""*, se necessário, como na imagem abaixo:





```
# If you change this file, run 'update-grub' afterwards to update
# /boot/grub/grub.cfg.
# For full documentation of the options in this file, see:
#   info -f grub -n 'Simple configuration'

GRUB_DEFAULT=0
#GRUB_HIDDEN_TIMEOUT=0
GRUB_HIDDEN_TIMEOUT_QUIET=true
GRUB_TIMEOUT=10
GRUB_DISTRIBUTOR=`lsb_release -i -s 2> /dev/null || echo Debian`
GRUB_CMDLINE_LINUX_DEFAULT="quiet splash"
GRUB_CMDLINE_LINUX=""

# Uncomment to enable BadRAM filtering, modify to suit your needs
# This works with Linux (no patch required) and with any kernel that obtains
# the memory map information from GRUB (GNU Mach, kernel of FreeBSD ...)
#GRUB_BADRAM="0x01234567,0xfefefefe,0x89abcdef,0xefefefef"

# Uncomment to disable graphical terminal (grub-pc only)
#GRUB_TERMINAL=console

# The resolution used on graphical terminal
# note that you can use only modes which your graphic card supports via VBE
# you can see them in real GRUB with the command 'vbeinfo'
#GRUB_GFXMODE=640x480

# Uncomment if you don't want GRUB to pass "root=UUID=xxx" parameter to Linux
#GRUB_DISABLE_LINUX_UUID=true

# Uncomment to disable generation of recovery mode menu entries
#GRUB_DISABLE_RECOVERY="true"

# Uncomment to get a beep at grub start
#GRUB_INIT_TUNE="480 440 1"
```

## Conclusão

Durante o período de realização do trabalho foi possível aplicar os conhecimentos aprendidos na aula de Sistemas Operacionais B, que auxiliaram na hora de compilar o kernel Linux. Ocorreram alguns erros em algumas tentativas de compilação mas não foram muitos, algo que ajudou a diminuir os eventuais erros foi a aba “Help” do kernel que nos mostra uma descrição do módulo. Por fim, o menor tempo obtido foi o de 12 minutos, que acreditamos ser aceitável para a compilação do kernel.