Compiling the Linux kernel and creating a bootable ISO from it

Hirbod Behnam · Following
8 min read · Aug 4, 2022

There are a lot of tutorials around the internet which guides you to compile the Linux kernel from source, create the file system from BusyBox and then run it using QEMU. But, I wanted to create a bootable ISO from it so I can boot it on my own computer. Or maybe you got a shitbox and want to try latest Linux kernel on it. With that said, let's begin.

. . .

The Linux Kernel

At first, obviously, we want to compile the Linux kernel. To do so, go to https://kernel.org/ and grab the latest Linux kernel. At the time of writing this story, the latest Linux version is 5.19. So I need to run these commands to get it:

```
wget https://cdn.kernel.org/pub/linux/kernel/v5.x/linux-5.19.tar.xz
tar xf linux-5.19.tar.xz
cd linux-5.19.tar.xz
```

Before actually compiling the kernel, make sure that you have `make` to run the makefile. On Ubuntu I need to run the following command to get all packages I need for compiling the kernel.

```
apt install build-essentials flex libncurses5-dev bc libelf-dev
bison
```

Now we need to configure the kernel. To do so, at first run `make defconfig`. This will create the default config file for compiling Linux. Then you can use `make menuconfig` to enter a TUI to edit the config file. One thing that might be useful if you are willing to run this kernel on older devices is unchecking the 64 bit kernel. As the time of writing, you don't need to tackle with other options. The default config is good enough. (but nothing is stopping you from messing around it!)

You might also want to use the config file for your current distro. To use it, just copy one of the `/boot/config-xxx` files in the Linux kernel source root with the name of `.config` .

Now use `make -j $(nproc)` to compile the kernel with all of your cores. For me with a i7-4790K it took about 50 minutes. After compiling, you should see the`arch/x86/boot/bzImage`. If the file does not exists, check the make file log. For me, I had this error message:

```
make[1]: *** No rule to make target 'debian/canonical-certs.pem',
needed by 'certs/x509_certificate_list'.  Stop.
```

If you got this error, open the `.config` file with a text editor and find keys of `CONFIG_SYSTEM_TRUSTED_KEYS` and `CONFIG_SYSTEM_REVOCATION_KEYS and empty their values. [source] Then compile using make again.`

. . .

BusyBox

Next we want to use BusyBox to create a minimal file system. At first, get the source code and extract it. These are the commands which I used:

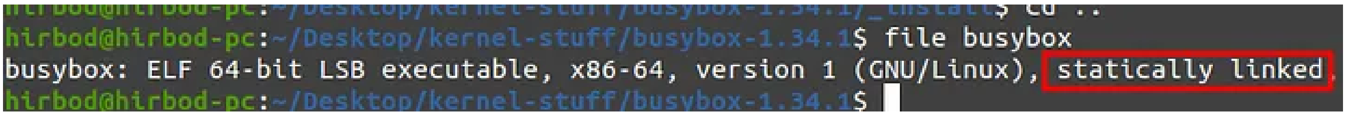
```
wget https://busybox.net/downloads/busybox-1.34.1.tar.bz2
tar xf busybox-1.34.1.tar.bz2
cd busybox-1.34.1
```

Just like Linux kernel, run `make defconfig` to make a default configuration for the BusyBox. Then use `make menuconfig` to bring the TUI and edit the configs. The option which you must edit is located in Settings, and then Build static binary (no shared libs). The reason that you want to enable this option is that we don't want to compile Glibc for our Linux distro.

Use `make -j $(nproc)` to compile the BusyBox. To check if the compiled file is fine, use this command:

```
file busybox
```

The file must be statically linked as shown in the picture below:



Info of busybox executable

Cross Compiling For 32-Bit Systems

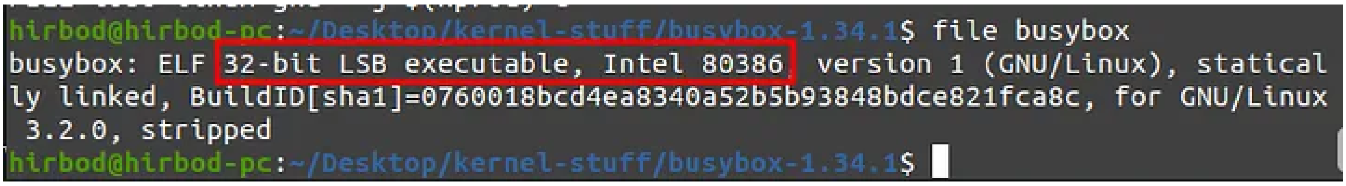
If you choose to compile the Linux kernel for 32-bit, you also want to compile the BusyBox for 32bit systems; To do so, you have to at first get the gcc i686 compiler using this command on Ubuntu:

```
apt install gcc-i686-linux-gnu
```

Now run the following commands to configure and compile the BusyBox:

```
make ARCH=i686 CROSS_COMPILE=i686-linux-gnu- defconfig # Generates the default config
make ARCH=i686 CROSS_COMPILE=i686-linux-gnu- menuconfig # Enable the static linking
make ARCH=i686 CROSS_COMPILE=i686-linux-gnu- -j $(nproc) # Compile BusyBox
```

Now lets run the `file busybox` again and see the results.

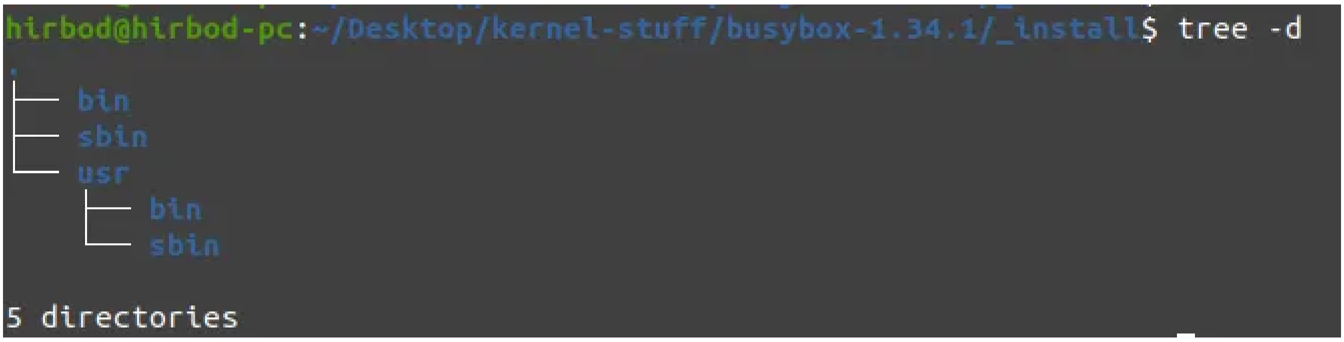


Info of busybox executable when cross compiled

As you can see, the binary is compiled in 32-bit. Note that it is still statically linked.

Creating The File System

At last, we have to create the file system which contains the BusyBox. At first, run `make install`. This will create a folder called `_install` which when you open it, you will see a hierarchy like Linux file system in it.



The files in _install folder

In this folder, run the following command to create the folders needed for kernel. (I'll explain a little why we need these.)

```
mkdir dev proc sys
```

Now create a file called `init` and open it with a text editor. Copy and paste the following data in it:

```
#!/bin/sh
mount -t devtmpfs none /dev
mount -t proc none /proc
mount -t sysfs none /sys
echo "Welcome to my Linux!"
exec /bin/sh
```

Then make the script executable with `chmod +x init` and we are done.

Let’s debrief what we just did; BusyBox made us one executable which is capable of providing us a lot of Linux utilities such as `sh`, `echo`, `vi` and so on. With `make install` we create a filesystem hierarchy which contains these programs as links to BusyBox executable. Next, we make a shell script called `init`. This script will be ran after kernel loads. At first, it mounts `dev`, `proc` and `sys` special directories. You can read about them from here and here. That’s why we had to create those directories! In the end, it prints a welcome message and runs `sh` to open a shell.

As a side note, you can put ANY executable as `init` file as long as it’s statically linked. You might also want to mount `dev`, `sys` and `proc` directories when the program starts using mount.h header. Read more about this here.

Last thing we need to do is to create the filesystem itself. To do so, run these commands inside _install directory.

```
find . -print0 | cpio --null -ov --format=newc | gzip -9 >
../initramfs.cpio.gz
```

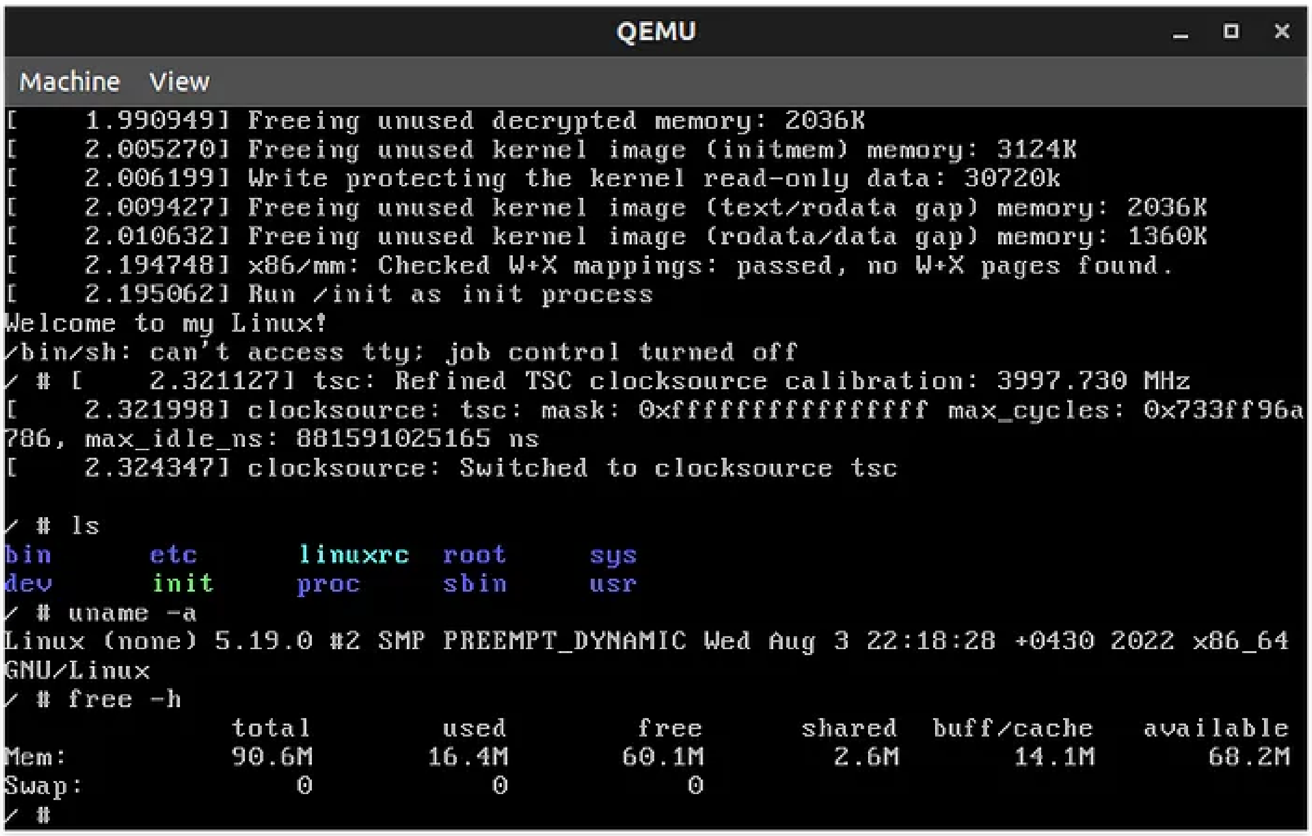
This will create the file `initramfs.cpio.gz` in upper directory.

. . .

Testing The Compiled Kernel With QEMU

Before creating the ISO file let us check if the kernel itself is fine or not. To do so, we use QEMU. Just run the following command: (make sure that you change the path of bzImage and initramfs)

```
qemu-system-x86_64 -kernel bzImage -initrd initramfs.cpio.gz
```



Running our Linux in QEMU

Now that we know that our kernel boots, let's create a bootable ISO for it.

. . .

Creating a Bootable ISO

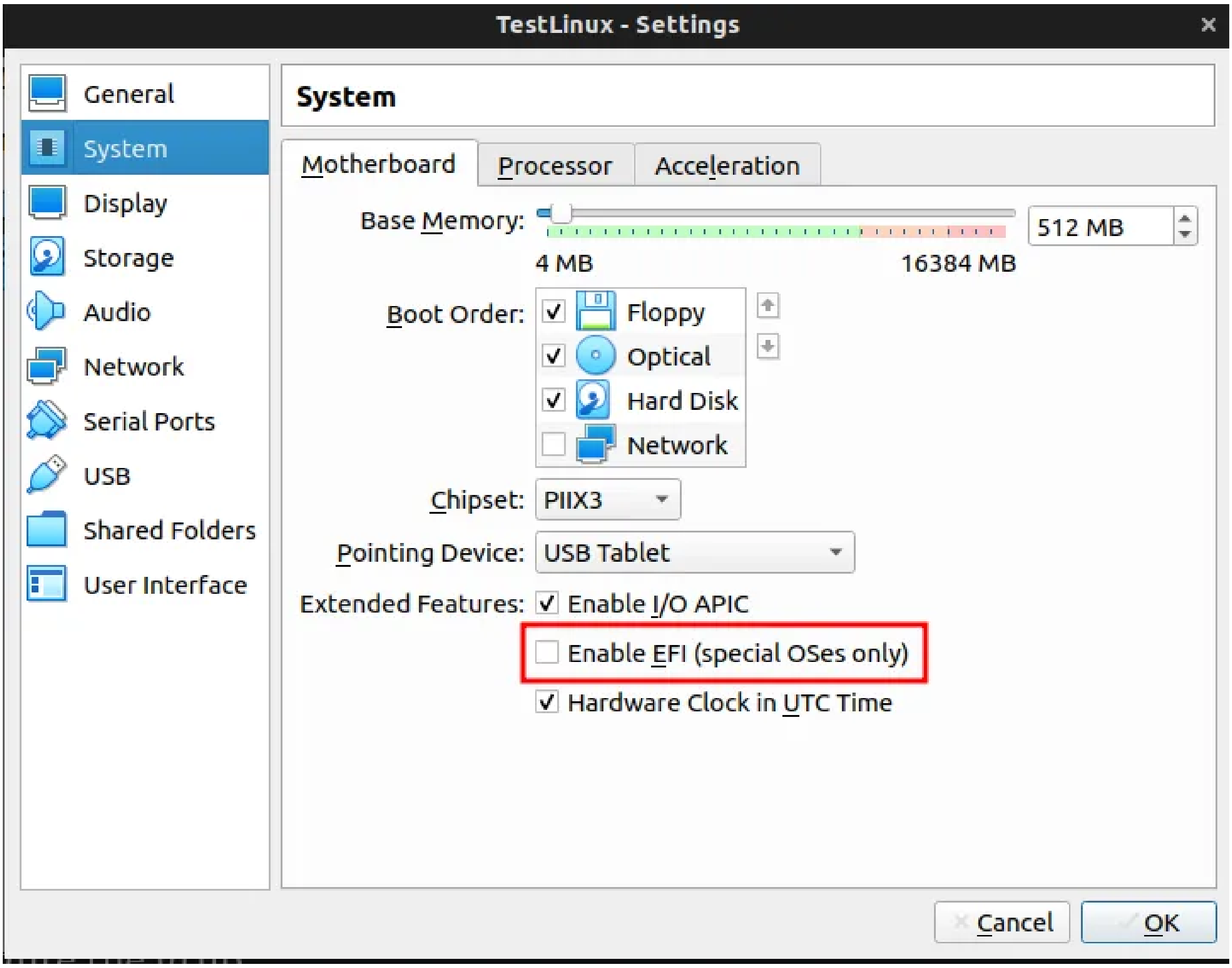
Preparing Files

Create a folder somewhere with any name you want. I name it `iso`. Then create a folder called `boot` in it and inside `boot` create a folder called `grub`. Then copy `bzImage` and `initramfs.cpio.gz` into boot folder.

Creating the Grub Config File

We will use `grub-mkrescue` to create our bootable ISO. But before doing so, we have to know if our current host is booted with UEFI or BIOS. To do so, check if the folder `/sys/firmware/efi` exists on your system or not. If it does, your computer uses UEFI otherwise it's BIOS.

So why knowing this is important? The `grub-mkrescue` uses the currently installed grub stuff to create the ISO image. This means that if your operating system is booted in BIOS, the chances are that the ISO created from `grub-mkrescue` does not support UEFI at all. In some cases, UEFI motherboards support booting BIOS images using CMS. But that's not always the case. If you want to make images for BIOS from UEFI host or vice versa, I suggest you to create a Debian virtual machine in VirtualBox. VirtualBox supports both BIOS and UEFI in it's motherboard settings. After choosing the appropriate one, install Debian (net install is sufficient) and move the folder which contains boot and grub folders to virtual machine. Then continue reading the guide to configure the grub and create the ISO file.



The location of switching between BIOS and UEFI in VirtualBox

Now we have to configure the grub itself. Create a file named `grub.cfg` in `grub` folder of the `boot` folder. If your host is booted using BIOS (and thus the output ISO is BIOS too) put these lines in the config file:

```
set default=0
set timeout=10

menuentry 'myos' --class os {
    insmod gzio
```

```
    insmod part_msdos
    linux /boot/bzImage
    initrd /boot/initramfs.cpio.gz
}
```

If you are using UEFI, put these lines in it:

```
set default=0
set timeout=10

# Load EFI video drivers. This device is EFI so keep the
# video mode while booting the linux kernel.
insmod efi_gop
insmod font
if loadfont /boot/grub/fonts/unicode.pf2
then
        insmod gfxterm
        set gfxmode=auto
        set gfxpayload=keep
        terminal_output gfxterm
fi

menuentry 'myos' --class os {
    insmod gzio
    insmod part_msdos
    linux /boot/bzImage
    initrd /boot/initramfs.cpio.gz
}
```

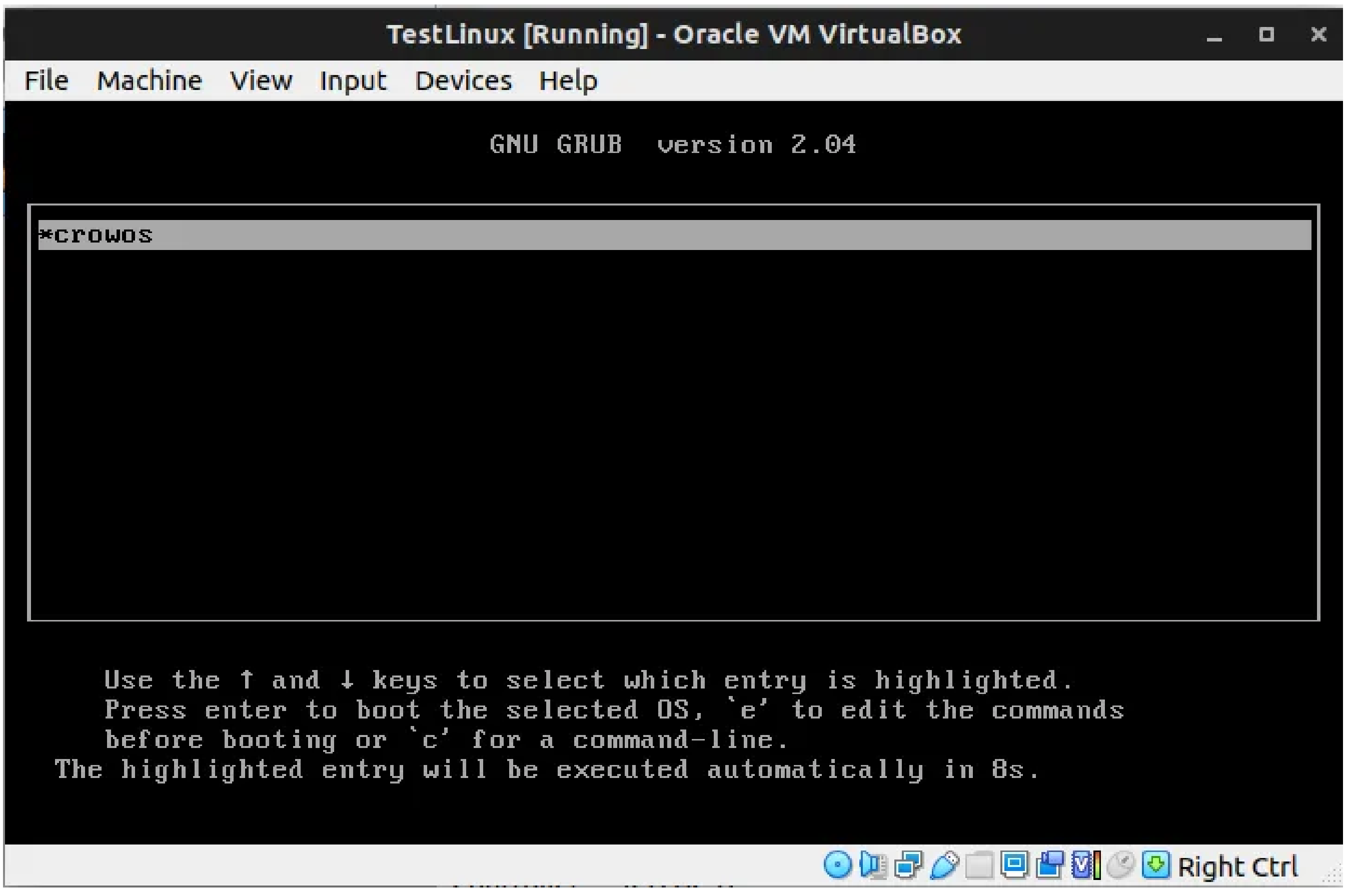
At last run this command to create the ISO file. Replace the last argument with the folder name which you created at first step.

```
grub-mkrescue -o myos.iso iso/
```

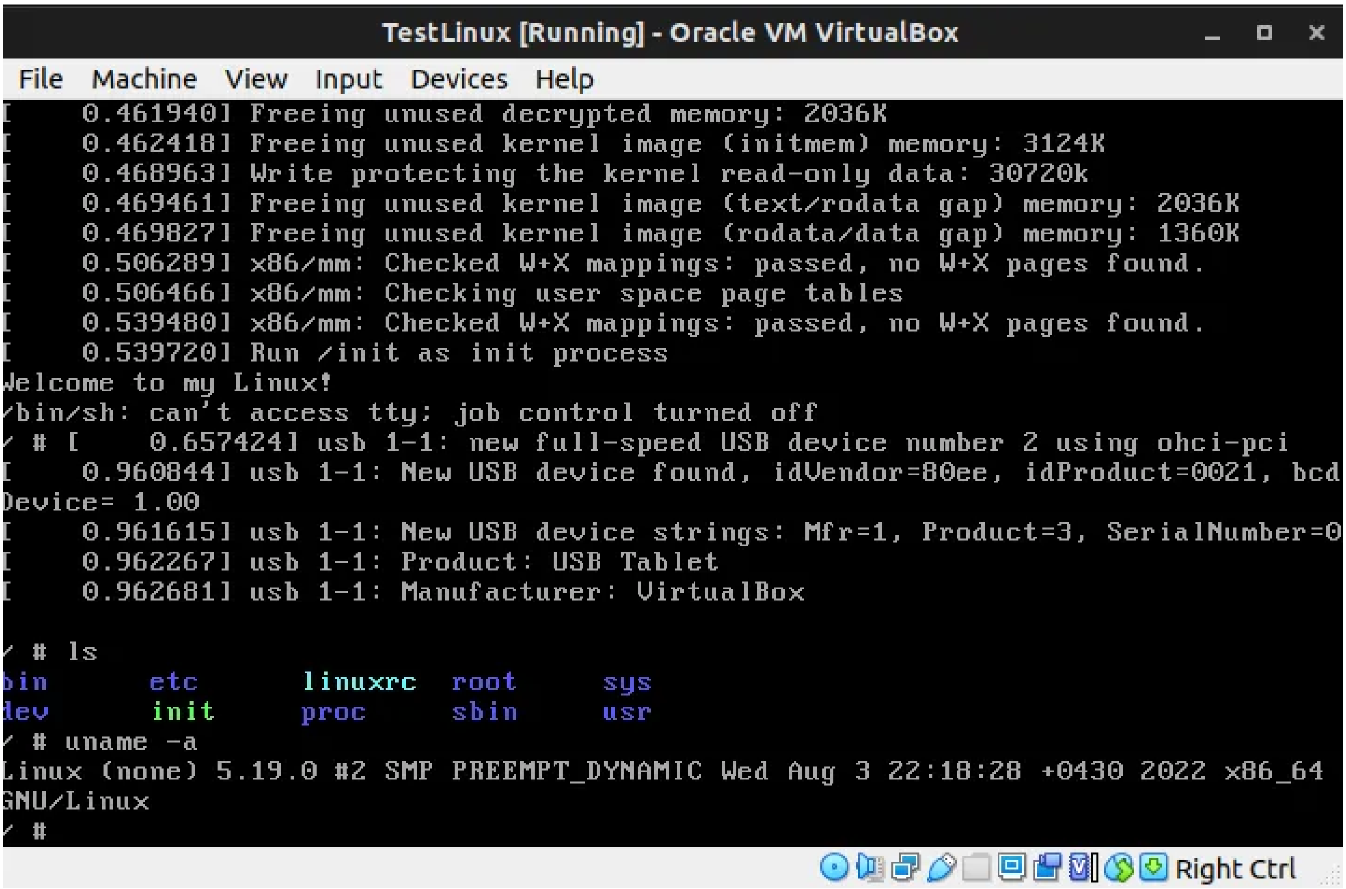
If you get *grub-mkrescue: error: xorriso not found*, simply install `xorriso` from your package manager.

Testing The ISO With VirtualBox

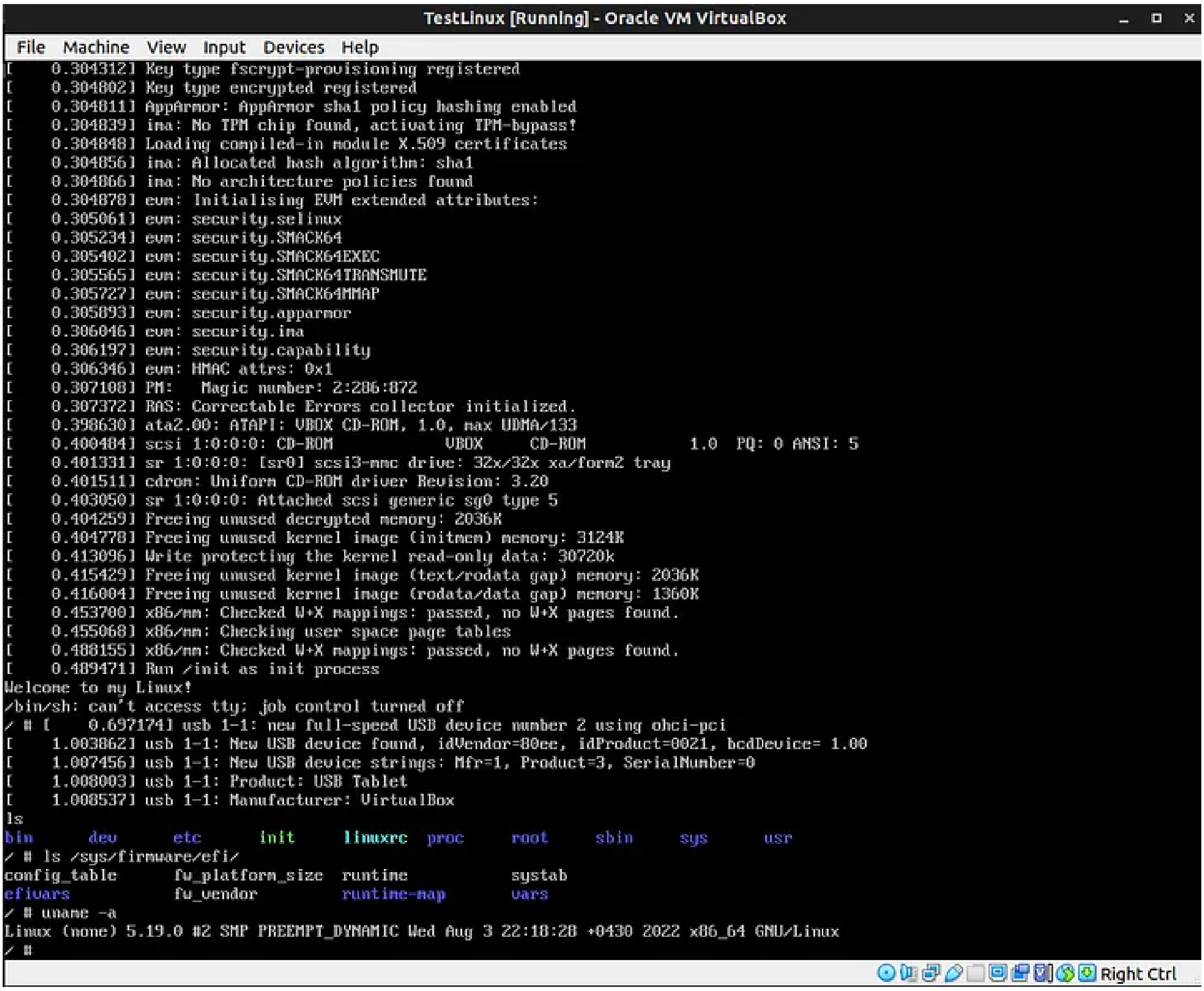
Before testing the ISO on a real computer, let us try booting from it in VirtualBox. To do so, create a new virtual machine and choose the ISO you have just created as the content of optical disk. Start the operating system. You must see the grub menu and then the operating system must boot. Don’t forget the select EFI in motherboard settings if needed.



Grub boot menu



Our kernel running in BIOS mode in VirtualBox



Our kernel running in UEFI mode in VirtualBox

Fix "error: kernel doesn't support 64-bit CPUs"

If you have compiled a 32bit kernel and tried to use Ubuntu to create a EFI ISO for it, you will probably come across this error message in grub. I'm not sure if this is a feature or a bug (see here) but you can simply use a Debian VM to create the ISO file to bypass this issue.

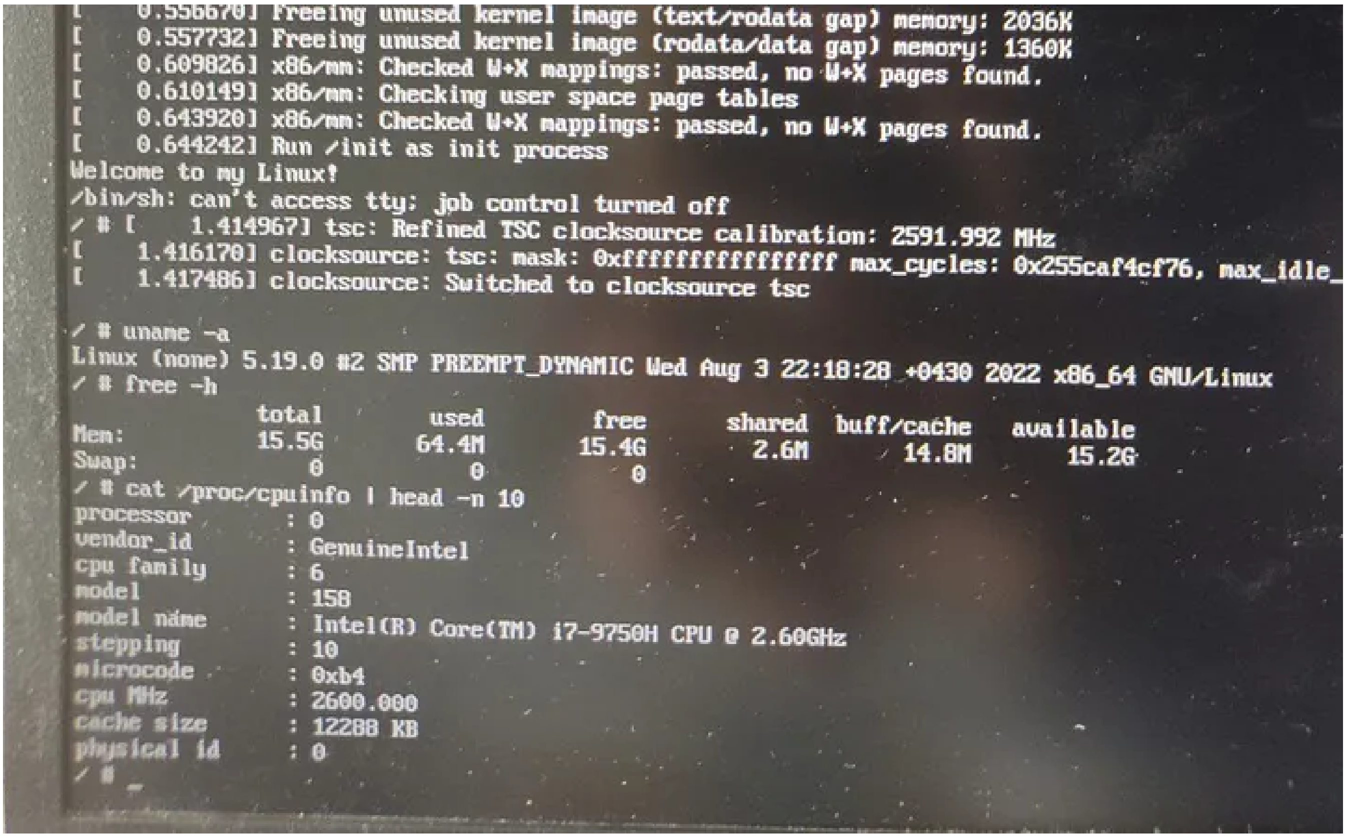
Fix Black Screen After Grub When Booting in UEFI

When booting in UEFI, after choosing your OS in grub, your screen might go blank and stay blank. This might indicate a problem in your kernel config. I suggest you check EFI frame buffer options and enable them.

Real Hardware Test

At the very end, we shall try our operating system on a REAL hardware. To do so, I personally use Rufus to create bootable flash drives. In the dialog which it asks if you want to write in DD or ISO mode choose DD mode.

Plug your flash into your laptop/PC, change the boot order and disable secure boot. Your OS should boot afterwards!



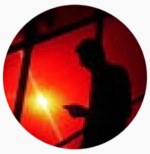
Linux kernel booted on real hardware

. . .

Sources

- https://gist.github.com/chrisdone/02e165a0004be33734ac2334f215380e
- https://subscription.packtpub.com/book/hardware-and-creative/9781783289851/1/ch01lvl1sec09/compiling-busybox-simple
- https://unix.stackexchange.com/a/238585/331589
- https://superuser.com/a/1509114/940438
- https://itsfoss.com/check-uefi-or-bios/
- https://askubuntu.com/a/1329625/746382
- https://www.reddit.com/r/Gentoo/comments/v1d36n/black_screen_after_grub_but_only_with_kernel/

Linux Kernel Linux Busybox Grub Bootable



Following

Written by Hirbod Behnam

7 Followers

More from Hirbod Behnam

Hirbod Behnam

NvChad on Debian netinst using xterm

Can you install NvChad on a bare Linux operating system which does not have window manager such as X11? We'll find out in this post!

5 min read · Feb 24

2

See all from Hirbod Behnam

Recommended from Medium

Arjun Pandey in Dev Genius

Entering into DevOps-26 Rust05

The fifth lesson in a series on the programming language Rust is now available. In this story we would be focusing on partial move. Please...

12 min read · May 18

4

Benoit Ruiz in Better Programming

Advice From a Software Engineer With 8 Years of Experience

Practical tips for those who want to advance in their careers

22 min read · Mar 21

8,704 174

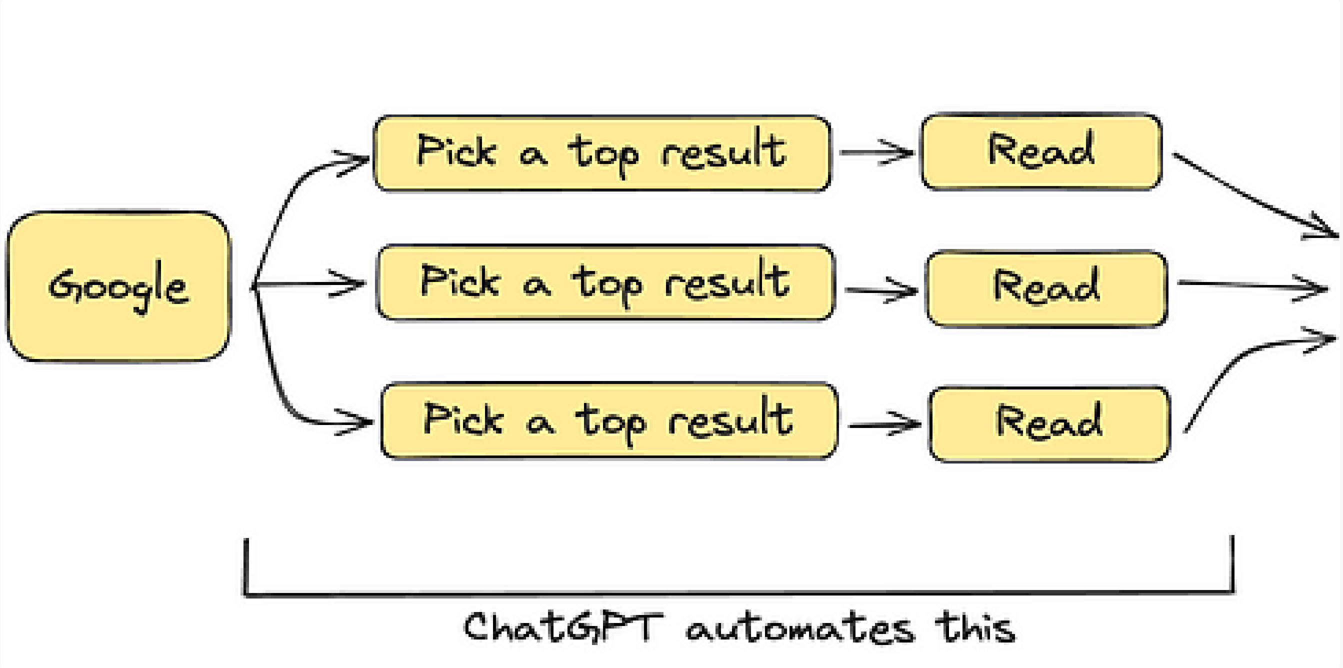
Lists

General Coding Knowledge

20 stories · 358 saves

Staff Picks

451 stories · 296 saves



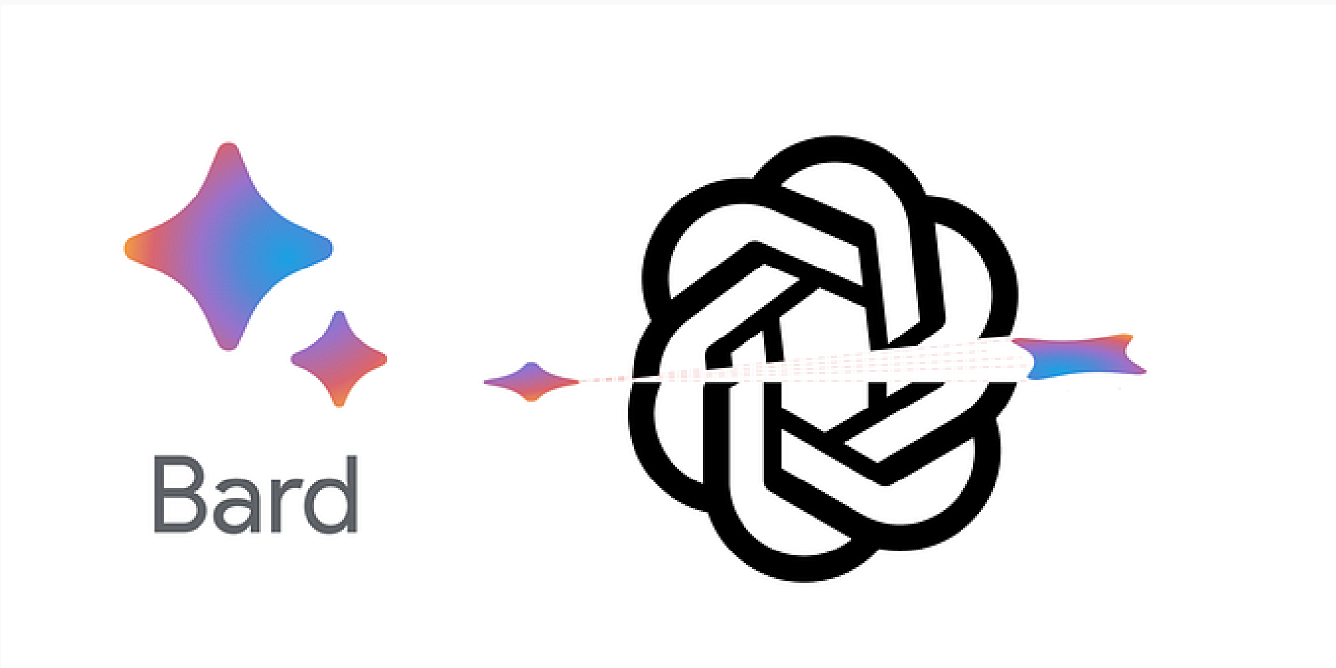
Elliot Graebert

How I replaced Google with ChatGPT

Incorporating AI into my daily engineering workflows.

10 min read · 3 days ago

255 8



AL Anany

The ChatGPT Hype Is Over — Now Watch How Google Will Kill ChatGPT.

It never happens instantly. The business game is longer than you know.

· 6 min read · Sep 2

8,507 258

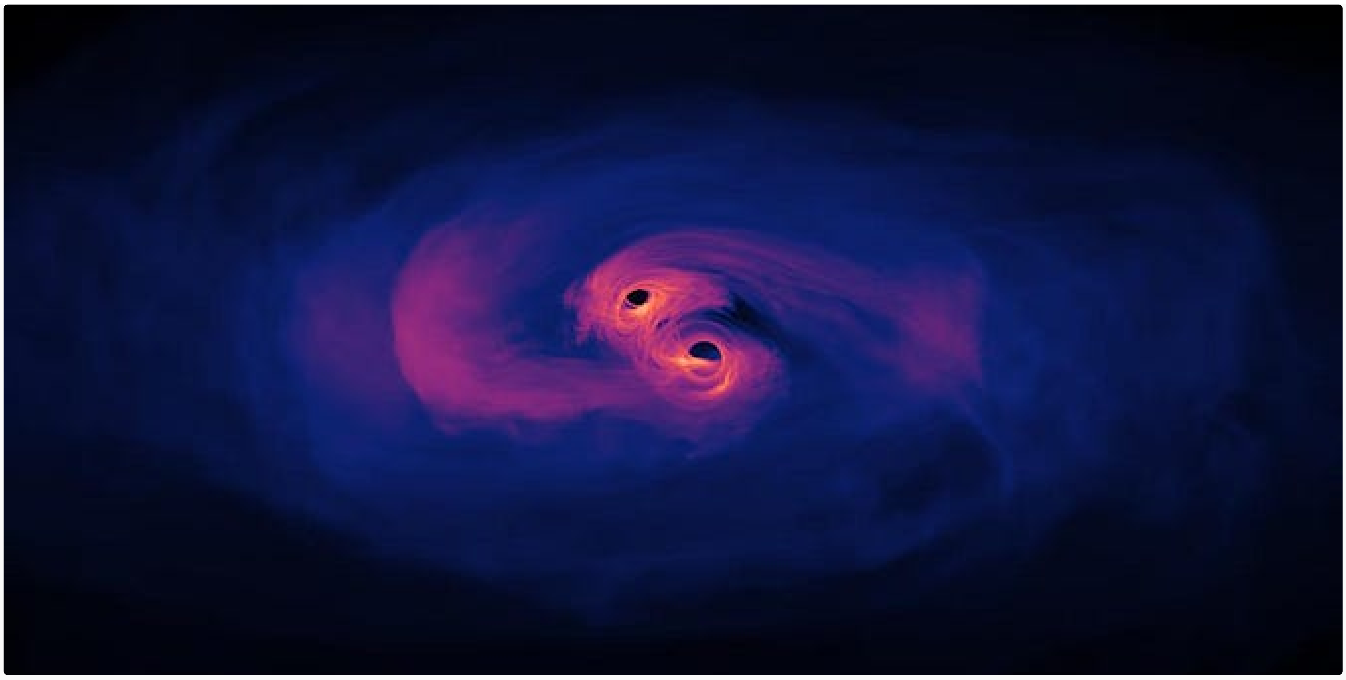
Tejesh Kumar in Towards Dev

Makefile: An Introduction

We all need things to be fast and easy. Faster CPU. Faster Memory. Faster Communication. Then, why is your C/C++ compilation still slow...

4 min read · Aug 13

55 2



Ethan Siegel in Starts With A Bang!

See more recommendations

Gravity doesn't happen instantly

Newton thought that gravitation would happen instantly, propagating at infinite speeds. Einstein showed otherwise; gravity isn't instant.

· 11 min read · Sep 14

1.3K 16