Kernel Build Guide

By: Isaiah Anyimi, Ona Igbinedion, Moriah Scott

Helpful Links

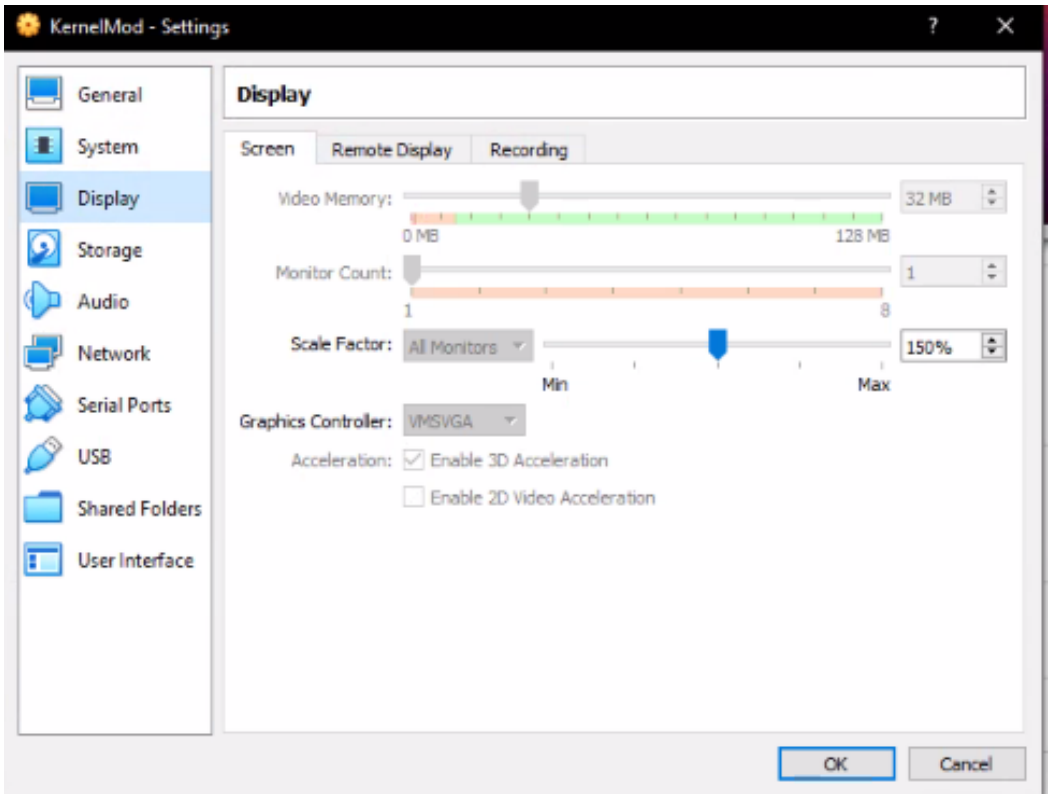
The links below were used when we were figuring out how to compile the kernel successfully, and boot it. The instructions have the solutions embedded within them, but if you would like to peruse through them, they have small references to what we used them for in the instructions.

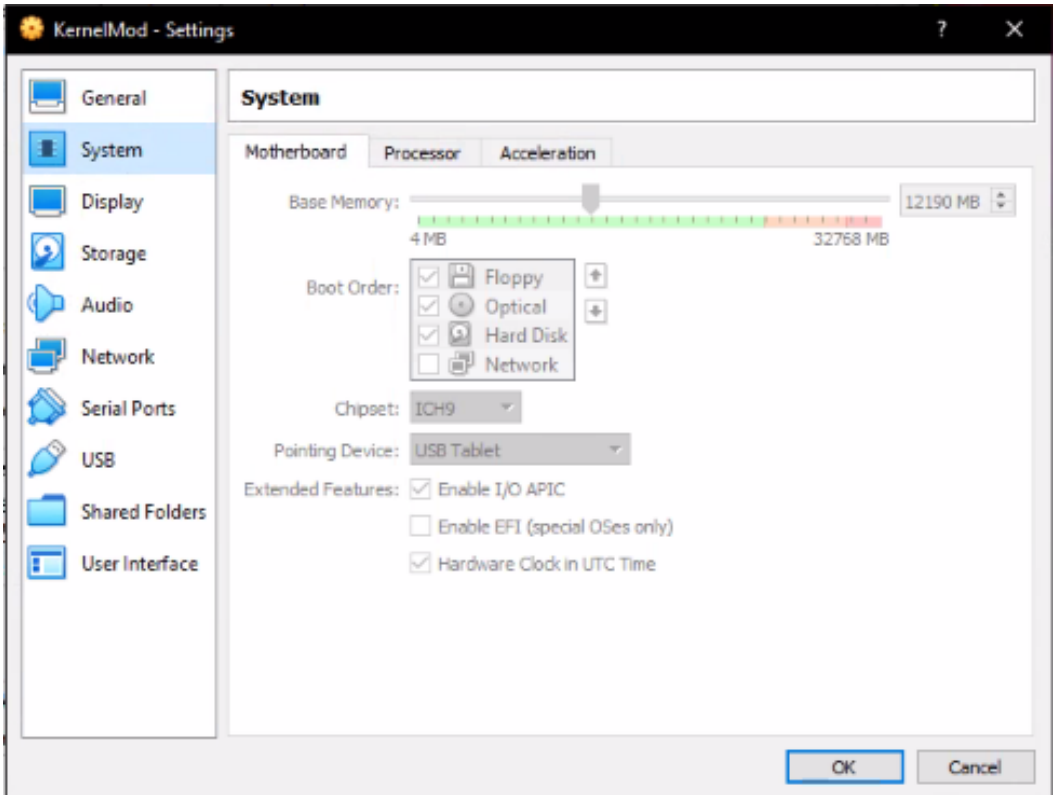
- https://forums.virtualbox.org/viewtopic.php?f=3&t=93990 (told us to regress VBox)
- https://forums.virtualbox.org/viewtopic.php?t=54926 (change chipset link)
- https://forums.virtualbox.org/viewtopic.php?f=1&t=62339 (hyper v)
- https://www.cyberciti.biz/tips/compiling-linux-kernel-26.html (how to compile kernel)
- https://www.wikihow.com/Become-Root-in-Linux (how to become root user)
- https://askubuntu.com/questions/56841/gpg-cant-check-signature (keyserver link)
- https://forums.virtualbox.org/viewtopic.php?t=82263 (guest additions part 2)
- https://www.configserverfirewall.com/windows-10/virtualbox-guest-additions-windows-10/ (guest additions part 1)
- https://unix.stackexchange.com/questions/293642/attempting-to-compile-kernel-yields-a-certification-error (comment out in .config)

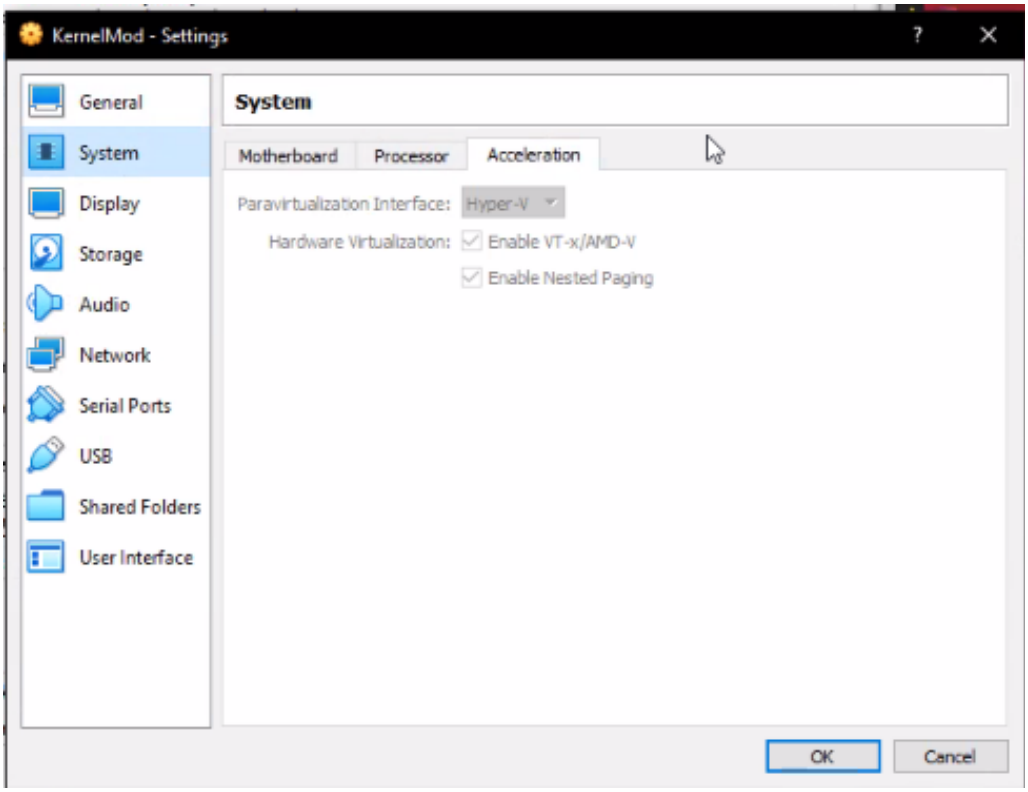
How we did a clean build of the Kernel

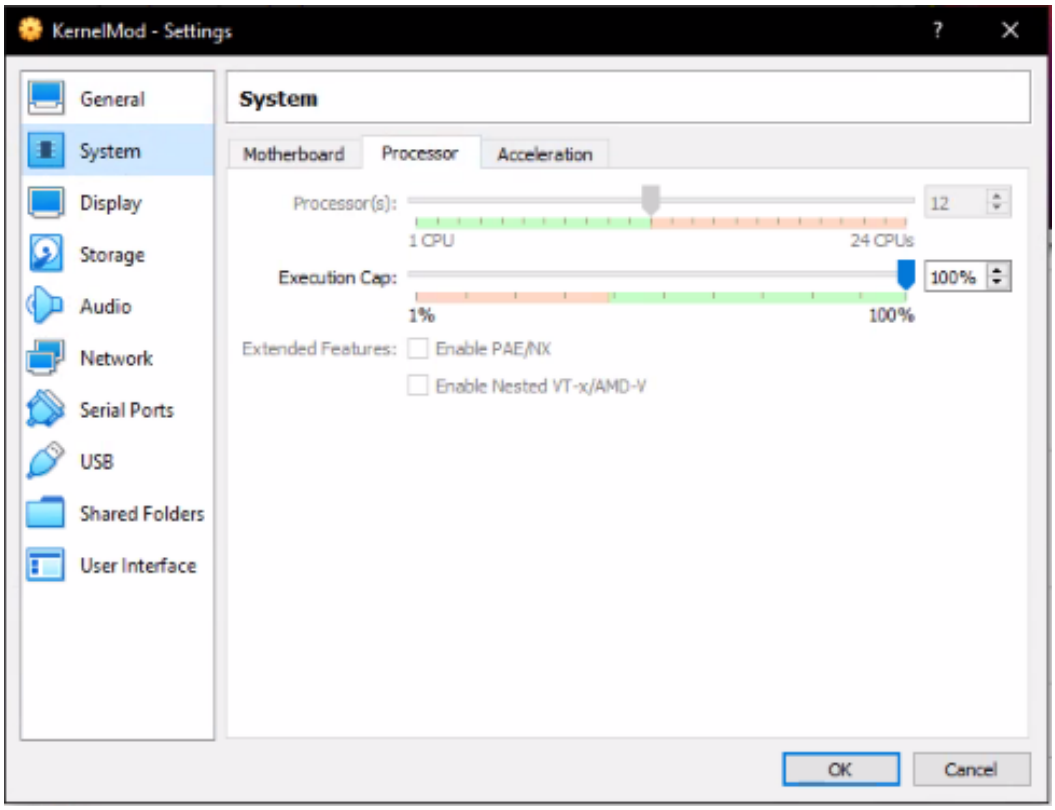
We followed the tutorial on https://www.cyberciti.biz/tips/compiling-linux-kernel-26.html to compile the kernel. We ran into several issues while doing so, but were able to debug them via existing VirtualBox forums.

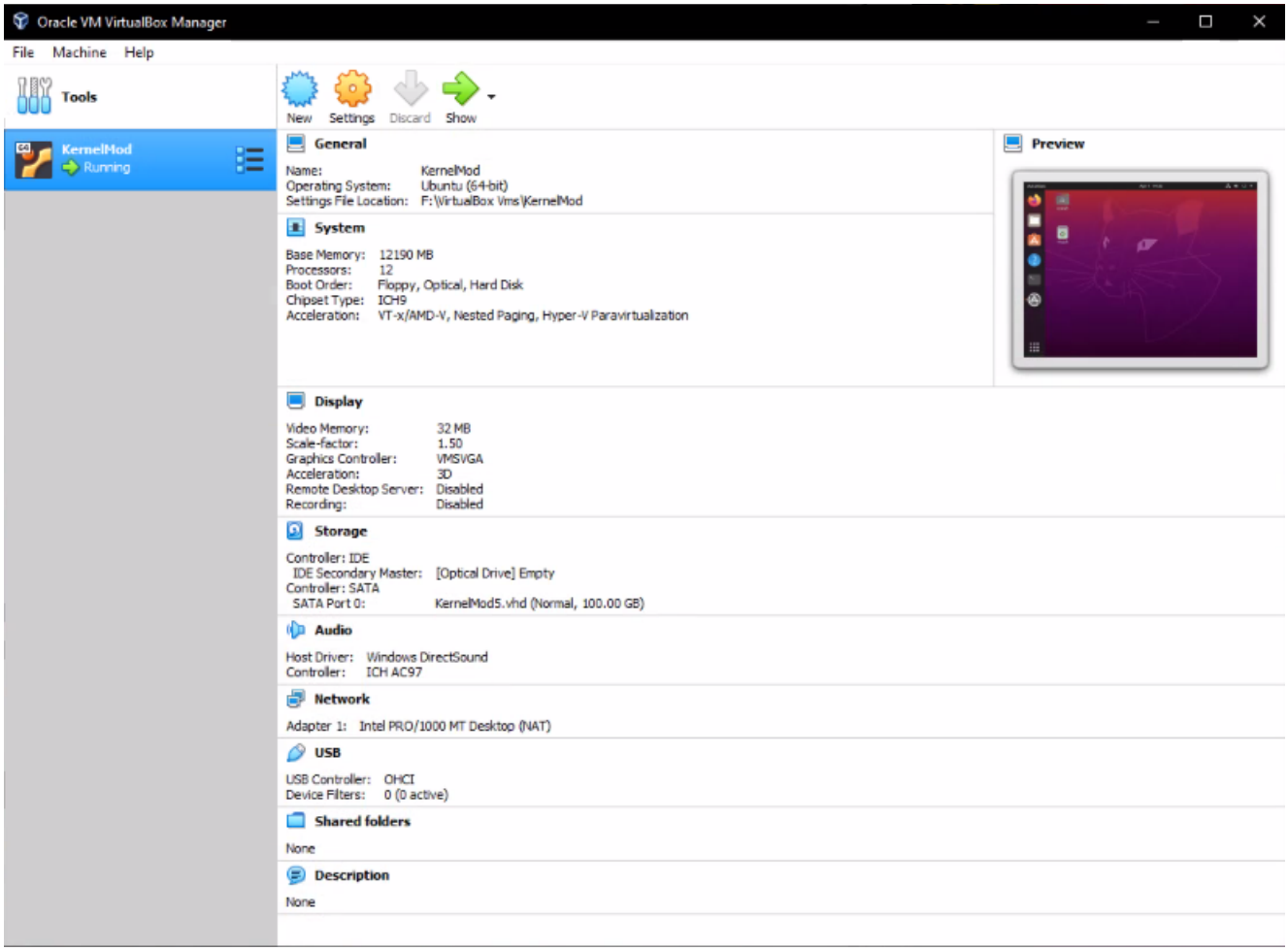
1. Install Oracle VM VirtualBox to version 6.0.6
    a. You can use the latest version of VirtualBox if you want, but if you are on a Windows computer be forewarned. Newer versions of VirtualBox were causing the error `Kernel panic - not syncing: Fatal exception` whenever we tried to run our build.
2. Set up your VM
    a. We had many issues setting up our VM. When building the kernel, we ran out of space when we set aside 40 gB for our virtual hard disk. We then set aside 100 gB and were able to compile successfully.
        b. Here are several pictures of our VM settings.


KernelMod - Settings
General
System
Display
Storage
Audio
Network
Serial Ports
USB
Shared Folders
User Interface
Display
Screen
Remote Display
Recording
Video Memory:
32 MB
0 MB
128 MB
Monitor Count:
1
1
8
Scale Factor:
All Monitors
150%
Min
Max
Graphics Controller:
VMSVGA
Acceleration:
Enable 3D Acceleration
Enable 2D Video Acceleration
OK
Cancel


KernelMod - Settings
General
System
Display
Storage
Audio
Network
Serial Ports
USB
Shared Folders
User Interface
System
Motherboard
Processor
Acceleration
Base Memory:
12190 MB
4 MB
32768 MB
Boot Order:
Floppy
Optical
Hard Disk
Network
Chipset:
ICH9
Pointing Device:
USB Tablet
Extended Features:
Enable I/O APIC
Enable EFI (special OSes only)
Hardware Clock in UTC Time
OK
Cancel


KernelMod - Settings
General
System
Display
Storage
Audio
Network
Serial Ports
USB
Shared Folders
User Interface
System
Motherboard
Processor
Acceleration
Paravirtualization Interface:
Hyper-V
Hardware Virtualization:
Enable VT-x/AMD-V
Enable Nested Paging
OK
Cancel


KernelMod - Settings
General
System
Display
Storage
Audio
Network
Serial Ports
USB
Shared Folders
User Interface
System
Motherboard
Processor
Acceleration
Processor(s):
12
1 CPU
24 CPUs
Execution Cap:
100%
1%
100%
Extended Features:
Enable PAE/NX
Enable Nested VT-x/AMD-V
OK
Cancel



c. We also used 12 cores for our VM, so the build would go by quickly (It took like an hour and a half-ish)
d. In settings → systems → Motherboard: change the chipset setting from **PIIX3 to ICH9**
e. In settings → systems → Acceleration: change Paravirtualization Interface to **Hyper-V**
f. In setting → systems → Acceleration: make sure enable **VT-x/AMD-V** is enabled. **You may have to go into the uefi firmware of your windows machine and toggle this setting as well.** It may be labeled as “Virtualization” or “SVM”. We found it in the overclocking settings.
g. In settings → display: click to **Enable 3D Acceleration**
h. In your VM, with the Ubuntu version of your choosing (our initial version was 20.0.4) go to terminal and make yourself the root user.
i. If you don’t know the root password (highly unlikely), enter the command `sudo passwd root` to set the root password to a password of your choosing. Then enter the command `su -` to become the root user. Enter the root password (you just created it!) and then you will become the root user

3. Navigate to the /usr/src directory (`cd /usr/src`) and download the linux kernel source code using the following command
    a. `wget https://cdn.kernel.org/pub/linux/kernel/v5.x/linux-5.11.10.tar.xz`
    b. You can replace the 5.11.10 with whatever kernel version you want.
4. Extract the tar.xz file
    a. Type in either of the following commands (We used the first one)
    b. `$ unxz -v linux-5.11.10.tar.xz` or `$ xz -d -v linux-5.11.10.tar.xz`
5. Verify Linux kernel tartball with pgp
    a. First grab the PGP signature for linux-5.11.10.tar using the following command:
        i. `$ wget https://cdn.kernel.org/pub/linux/kernel/v5.x/linux-5.11.10.tar.sign`
    b. Try to verify it with the command `$ gpg --verify linux-5.11.10.tar.sign`
        i. You should get a sample output like the following

```
gpg: assuming signed data in 'linux-5.6.9.tar'
gpg: Signature made Sun 12 Aug 2018 04:00:28 PM CDT
gpg:                using RSA key 79BE3E4300411886
gpg: Can't check signature: No public key
```

    c. Grab the public key from the PGP keyserver in order to verify the signature i.e. RSA key ID 79BE3E4300411886 (from the above outputs) using the following commands
        i. `gpg --keyserver keyserver.ubuntu.com --recv-keys 79BE3E4300411886` and `gpg --no-default-keyring -a --export 79BE3E4300411886 | gpg --no-default-keyring --keyring ~/.gnupg/trustedkeys.gpg --import`
        ii. We used both of these commands, because when we used the command in the original tutorial, it didn't let us grab the public key and gave us an error. This resolved it!
    d. Now verify gpg key again with the gpg command: `$ gpg --verify linux-5.11.10.tar.sign`
        i. You should get a sample output like the following

```
gpg: assuming signed data in 'linux-5.6.9.tar'
gpg: Signature made Sun 12 Aug 2018 04:00:28 PM CDT
gpg:                using RSA key 79BE3E4300411886
gpg: Good signature from "Linus Torvalds <torvalds@kernel.org>" [unknown]
gpg:                 aka "Linus Torvalds <torvalds@linux-foundation.org>" [unknown]
gpg: WARNING: This key is not certified with a trusted signature!
gpg:          There is no indication that the signature belongs to the owner.
Primary key fingerprint: ABAF 11C6 5A29 70B1 30AB  E3C4 79BE 3E43 0041 1886
```

   ii. If this doesn't work(You get a "BAD signature" output), RIP, then search the forums with your error and you should find a solution eventually. If not, post your issue to the forums and some people may help you eventually. If it does work, move on to the next step.

e. untar/extract the Linux kernel tarball using the tar command below
   i. `$ tar xvf linux-5.11.10.tar`

6. Configure the Linux kernel features and modules:
   a. Move into the newly created linux-5.11.10 directory
      i. `$ cd linux-5.11.10`
   b. We copied an existing .config file using the cp command, because that is easiest!
      i. `$ cp -v /boot/config-$(uname -r) .config`
      ii. Sample output below:

```
'/boot/config-4.15.0-30-generic' -> '.config'
```

7. Install the required compilers and other tools
   a. Type in the following command to install them:
      i. `$ sudo apt-get install build-essential libncurses-dev bison flex libssl-dev libelf-dev`
8. Configuring the kernel
   a. Open the .config file and edit the following:
      i. Comment out `CONFIG_SYSTEM_TRUSTED_KEYS` and `CONFIG_MODULE_SIG_KEY` and save the file.
   b. Type in one of the following options to start kernel configuration
      i. `$ make menuconfig` (We used this way)
         1. Navigate to Load and load whatever additions you want to add (optional)
         2. Navigate to Save and save .config file (not optional!)
      ii. `$ make xconfig`
      iii. `$ make gconfig`
9. Compile the kernel

    a. If you added additional cores run the following command, replacing the 12 with how many cores you added
        i. `$ make -j 12`
    b. If you did not add additional cores run the following command
        i. `$ make`
10. Install the Linux Kernel and Kernel Modules
    a. Install Linux Kernel Modules using `$ sudo make modules_install`
    b. Install Linux Kernel using `$ sudo make install`
11. Reboot your vm by issuing the `# reboot` command and hope it loads properly.