NCS 490:
Compiling from Source
and
Building a Kernel

By: Tony Perez

February 9, 2015

Abstract

For this this lab we learned how to compile software from source code and saw how the kernel was setup and built. Both tasks were more tedious then a simple install of a software or just installing the default kernel. Both compiling from source and building a kernel will help to target the specific CPU and hardware specs and so it would run smoother and also be more light weight since we would only be installing what we want. Although it is not necessary to do this all the time it is a good practice for when we need to built the best system for a specific task and would want it to run well.

1 Introduction

We first looked at how to compile software from source by installing a game called Net Hack. Compiling from source is different then just installing a file using yum. When you compile from source the program would actually be optimized for the system because it built it itself and so it would be more native. Also you could install it with different flags and so you could pick what get installed. For example you could want just the core program, some features or all of the features to be included. Building and compiling the kernel from source works the same way. It makes a clean install and you would choose what you want. There are no default setting unless you leave it that way. You install the drivers, packages, and it is optimized to work with your current processor. Both will be more efficient and run smoother because they are built for your specific system, if supported.

2 Compiling From Source: Installing Net Hacks

We were assigned to install a game called Net Hacks.
We first started of by installing the *Development Tools* package group and the *ncurses-devel* package using the yum command.

```
# yum groupinstall Development Tools
```

The *groupinstall* install multiple packages within *Development Tools*.
We then downloaded the Net Hack source code from here and downloaded the *nethack-343-src.tgz* file to our systems using the **wget** command

```
# wget http://softlayer-dal.dl.sourceforge.net/project/nethack/nethack/3.4.3/nethack-343-src.tgz
```

We extracted the source code into the directory */usr/src/nethack/*. We then headed over to Net Hack Wiki to see how to compile it.

```
# tar -xvf nethack-343-src.tgz -C /usr/src/nethack/
```

We then went into the directory we extracted it to and ran the setup script

```
# cd /usr/src/nethack/nethack-3.4.3/
# sh sys/unix/setup.sh x
```

Edited *include/unixconf.h*. We removed the */* */* that were around *#define LINUX.*. Then we edited the

```
40 /* #define LINUX */     /* Another Unix clone */
```

```
40 #define LINUX          /* Another Unix clone */
```

src/Makefile and uncomminted the *WINTTYLIB = -ltermlib* line by removing the #.

```
233 # WINTTYLIB = -lncurses
```

```
233 WINTTYLIB = -lncurses
```

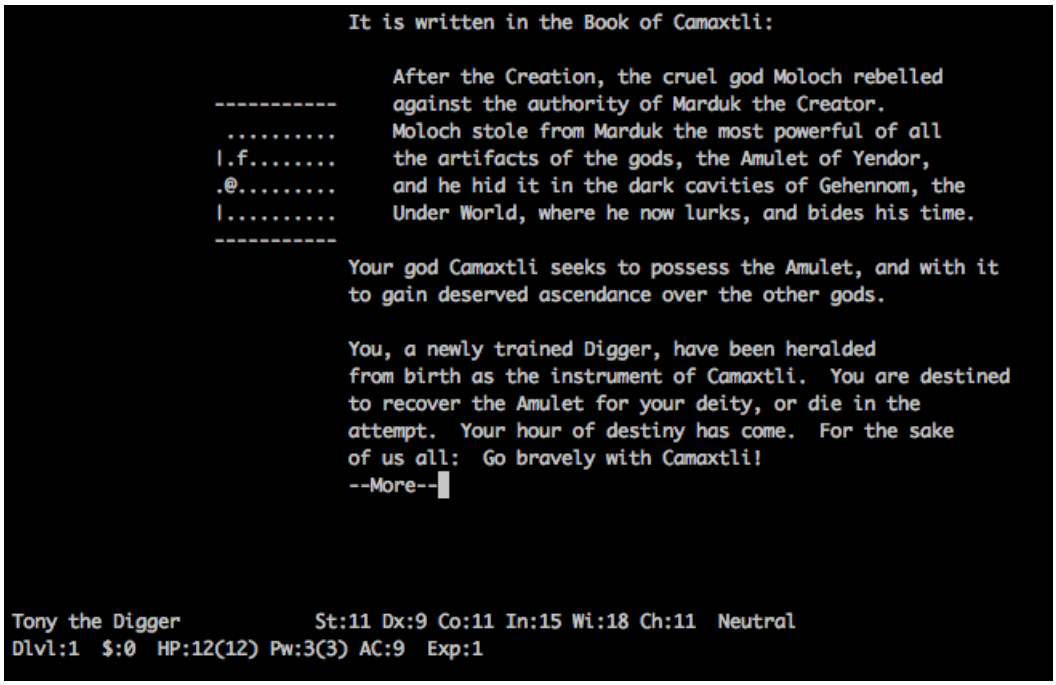
Finally to compile the program we ran the *make install* command within the Net Hack directory and added our user to the game group to be able to play it

```
# make install
# usermod -a -G games tony
```

Next we go into our games directory and run nethack.

```
# cd /usr/games
# ./nehack
```

You will get this:



3 Building and Compiling a Kernel

In this part of the lab we looked at compiling a kernel from source. The reason we looked at this is because this method shows us that we can make a kernel with only the drivers and setting we want. Most kernels would have many drivers and modules for general use and this is good if we want a quick setup but if we want the most efficient and a clean install you would hand pick the things you needed.

To more information about our system we installed some tools that will make it easier to view the.

```
# yum install hmaccalc zlib-devel binutils-devel elfutils-libelf-devel pciutils
```

We then went to hrefhttps://www.kernel.org/kernel.org to get the link for the latest kernel source code. We then used *wget* to download the .xz file.

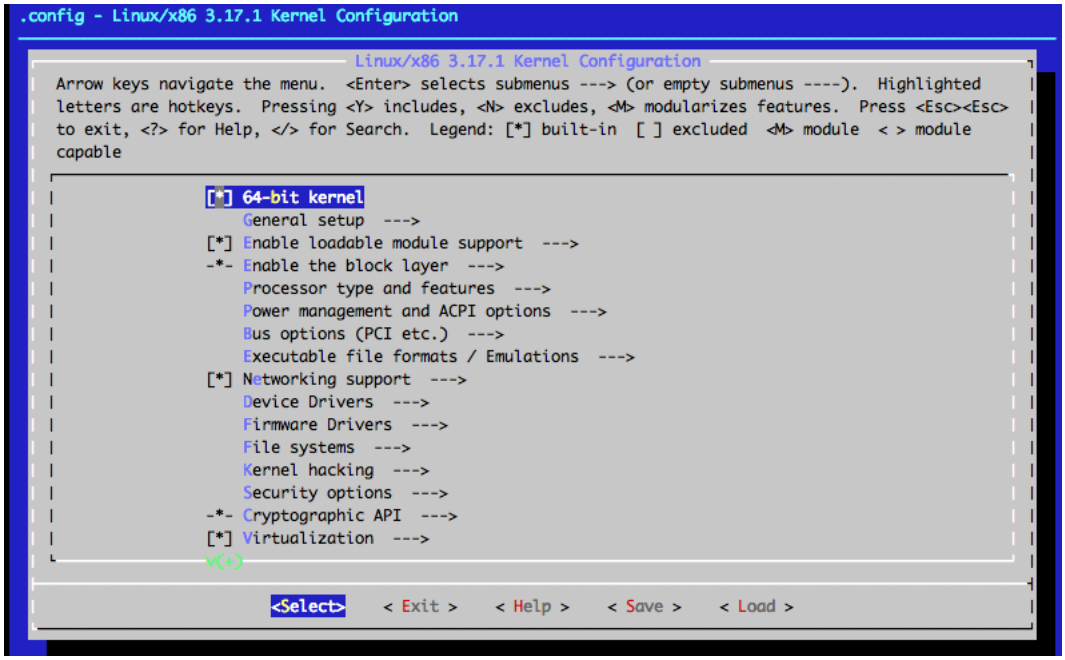
```
# tar xvfJ linux-3.17.1.tar.xz
```

We use the *lspci* command which lists all PCI devices.

```
[root@pereztr-1 games]# lspci
00:00.0 Host bridge: Intel Corporation 440FX - 82441FX PMC [Natoma] (rev 02)
00:01.0 ISA bridge: Intel Corporation 82371SB PIIX3 ISA [Natoma/Triton II]
00:01.1 IDE interface: Intel Corporation 82371SB PIIX3 IDE [Natoma/Triton II]
00:01.2 USB controller: Intel Corporation 82371SB PIIX3 USB [Natoma/Triton II] (rev 01)
00:01.3 Bridge: Intel Corporation 82371AB/EB/MB PIIX4 ACPI (rev 01)
00:02.0 VGA compatible controller: Cirrus Logic GD 5446
00:03.0 SCSI storage controller: XenSource, Inc. Xen Platform Device (rev 01)
```

This shows what devices we are being used so when we are going through the kernel we now what to include. Using the *make menuconfig* command within the *linux-3.17.1* directory we extracted.

```
# cd linux-3.17.1/
# make menuconfig
```



Here we can configure and only install only what we need like the type of CPU, file systems, drivers, and many more options. There are many options and would take a while to go through all the settings to only get the things your system needs. Usually the default options for the processor support both Intel and AMD. For hardware we could choose what we have like the type of network card we use. Instead of supporting a

wide variety of hardware we could choose what we need. This would optimize boot time, be more secure, and improve performance.
Once you are done hand picking what you really need, to compile you run this command:

```
# make -j2 && make modules_install
```

Depending on what you choose to install it could be quick or long. The less extra stuff you used the faster it will compile.

4 Conclusion

When compiling from source it benefits us because it tells the machine what it needs to run it. When using the generic way you get a lot of extra stuff because it is meant to target many type of systems. Compiling form source is tedious but when you really want to optimize this how it is done.