Sistemas Operacionais B

Atividade #1

1. Introdução

Esta atividade deverá permitir ao aluno familiarizar-se com o processo de compilação de um kernel Linux. Espera-se que ao final da atividade, cada aluno seja capaz de configurar, compilar, instalar e testar um novo kernel Linux. É importante entender o funcionamento de todos os passos envolvidos no processo de compilação do kernel, familiarizando-se com os comandos e ferramentas utilizados, o tempo necessário para cada passo, bem como os possíveis erros que podem ocorrer em cada etapa e as possíveis soluções, uma vez que tais passos serão realizados repetidas vezes ao longo do curso.

Esta atividade é baseada no material utilizado no curso Linux Kernel & Device Driver Programming (COP5641) Spring 2013 ministrado pela Dr. Sarah Diesburg na The Florida State University (http://www.cs.uni.edu/~diesburg/courses/dd/index.html).

2. Descrição da atividade

Summary:

This is not exactly a programming exercise, since you will not be writing any new code. You will step through the mechanical steps of configuring, compiling, installing, and testing a kernel. This will make sure you can handle the mechanics, let you see first-hand the time scales of some of the development steps you will be repeating frequently throughout the course, and expose you to some of the many possible failure modes of a kernel module.

Objectives:

- Learn a few Linux system administration commands, the bare minimum you need to compile, install, and test a kernel.
- Learn to reduce boot time and compile time by turning off unneeded Linux system services, and by pruning kernel options.
- Experience configuring, compiling, and installing a Linux kernel.
- Experience compiling and installing a set of Linux kernel modules.
- Experience a kernel crash.

Tasks:

Perform the following, on the machine in the lab or on a machine of your own that you can bring in to demonstrate.

These instructions are for use with a particular Linux system distribution (Ubuntu 16.04 LTS). If you are using a different Linux distribution you will need to adapt the instructions, or install a compatible distribution.

The detailed instructions explain one way of doing the job. If you have built Linux kernels before, you are free to compile the kernel in your own way, so long as it works.

If you are using a machines in the lab or your own machine, I recommend:

- Be prepared for the possibility that your machine's contents may be corrupted, requiring you to reload the entire system, possibly losing whatever else you had on the system.
- Consider setting up your system in a Virtual Machine (VM). This can allow you to run Linux in a separate environment from your normal OS installation, reducing the chance that you will corrupt your other system.
- Avoid partitioning your disk in a way that relies on using Logical Volume Manager (LVM).
- Separate your personal files (in /home) from the OS files (in /), using different disk partitions, so that if you need to reload the OS you will not need to overwrite your home directory.
- Use a Linux distribution that has a kernel, no later than 4.x.

User Accounts Setup

Power on and boot up the machine, if it is not already running, and log in as "root". If you did not do so previously, create yourself at least one ordinary user account, so that you don't have to do all your work as "root". To add a user account to the system perform the following steps:

- sudo adduser partner will create a new user named partner and setup the associated /home/partner directory structure.
- sudo adduser partner sudo will add the user you created above (partner in this example) to the sudoers list. This, effectively, gives partner root access.

I recommend that you only do as root those things that require it, to reduce the risk of clobbering your system if you make a mistake. In general, it is wise to log in first as a normal user, and then just "su" to root temporarily when necessary, or to use one of the Linux "virtual consoles" for a separate root login that you use just for the commands that require it. Yet another options is to use sudo to run individual commands as root.

Minimal Services

If you sitting in front of the computer at the physical console, you can select any of six virtual consoles using the function keys F1-F6 + CTRL + ALT. F1 is the initial virtual console. F7 is reserved for the X-windows display. For example, if you are running X-windows, you can switch over to virtual console #2 using the F2 + CTRL + ALT key to log in as root and do something, then switch back to the X-windows display using F7 + CTRL + ALT.

Assuming you performed the default installation, the machine will probably come up with an X-windows display manager screen. If so, edit the file */etc/init/rc-sysinit.conf* to change the initial run-level from 2 to 1 edit the line to be *env DEFAULT_RUNLEVEL=1*. You can get X-windows back later by changing the runlevel from 2 to 5, of by just executing the startx command from a login shell.

Shut off all unnecessary system services. The objective here is to allow your machine to boot quickly, and to free up as much as possible of the memory so that your kernel compilations will go as quickly as possible.

You will need to exercise a little bit of trial and error, and some web searching on the names of the services, to see which ones you can do without.

Test the system with the reduced services, by rebooting, and verifying that the system is working well enough for you to continue with the rest of this exercise. You can use the command "reboot" to reboot.

The Linux Kernel

Get a copy of the Linux kernel sources. You can download them from **http://www.kernel.org** or use apt-get source.

You may need to install additional deb (Debian package manager files) to compile and build a kernel. For Ubuntu 16.04 LTS, you need to install at least *libncurses5-dev*. Assuming you have networking up and running, you can do this using the apt-get and apt-cache utility. To install individual packages, do sudo apt-get install libncurses5-dev, etc. You can read more about apt-get and *apt-cache* in the man-page.

As the normal user to whose home directory you copied the kernel source code, extract the kernel source tree from the archive file and put in a location of your choice. For example, if you followed the steps above, you could next do the following:

```
cd /home/user
tar xjpf linux-4.xxx.tar.bz2
sudo ln -s `pwd`/linux-4.xxx /usr/src/linux
```

If your file is gzipped instead of bzip2ed, you use the tar option "z" instead of "j".

Once you have the kernel source tree uncompressed and untarred, cd into the source directory (*/usr/src/linux* in the examples above), and configure the kernel, as follows:

```
cd linux
make menuconfig
```

The kernel comes in a default configuration, determined by the people who put together the kernel source code distribution. It will include support for nearly everything, since it is intended for general use, and is *huge*. In this form it will take a very long time to compile and a long time to load. For use in this course, you want a different kernel configuration. The "make menuconfig" step allows you to choose that configuration. It will present you with a series of menus, from which you will choose the options you want to include. For most options you have three choices: (blank) leave it out; (M) compile it as a module, which will only be loaded if the feature is needed; (*) compile it into monolithically into the kernel, so it will always be there from the time the kernel first loads.

There are several things you want to accomplish with your reconfiguration:

- Reduce the size of the kernel, by leaving out unnecessary components. This is helpful for kernel development. A small kernel will take a lot less time to compile and less time to load. It will also leave more memory for you to use, resulting in less page swapping and faster compilations.
- Retain the modules necessary to use the hardware installed on your system. To do this without including just about everything conceivable, you need figure out what hardware is installed on your system. You can find out about that in several ways.

Before you go too far, use the "General Setup" menu and the "Local version" and "Automatically append version info" options to add a suffix to the name of your kernel, so that you can distinguish it from the "vanilla" one. You may want to vary the local version string, for different configurations that you try, to distinguish them also.

If you have a running Linux system with a working kernel, there are several places you can look for information about what devices you have, and what drivers are running.

- Look at the system log file, /var/log/messages or use the command dmesg to see the messages printed out by the device drivers as they came up.
- Use the command lspci -vv to list out the hardware devices that use the PCI bus.
- Use the command lsusb -vv to list out the hardware devices that use the USB.
- Use the command lsmod to see which kernel modules are in use.
- Look at /proc/modules to see another view of the modules that are in use.
- Look at /proc/devices to see devices the system has recognized.
- Look at /proc/cpuinfo to see what kind of CPU you have.
- Look at /proc/meminfo to see how much memory you have.

Using the available information and common sense, select a reasonable set of kernel configuration options. Along the way, read through the on-line help descriptions (for at least all the top-level menu options) so that you become familiar with the range of drivers and software components in the Linux kernel.

Before exiting the final menu level and saving the configuration, it is a good idea to save it to a named file, using the "Save Configuration to an Alternate File" option. By saving different configurations under different names you can reload a configuration without going through all the menu options again. Alternatively, you can backup the file (which is named .config manually, by making a copy with an appropriate name.

One way to reduce frustration in the kernel trimming process (which involves quite a bit of guesswork, trial, and error) is to start with a kernel that works, trim just a little at a time, and test at each stage, saving copies of the .config file along the way so that you can back up when you run into trouble. However, the first few steps of this process will take a long time since you will be compiling a kernel with huge number of modules, nearly all of which you do not need. So, you may be tempted to try eliminating a large number of options from the start.

Do the following *make* steps:

```
make
make modules_install
make install
```

The first command will compile the kernel and create a compressed binary image of the kernel. After the first step, the kernel image can be found at /boot/vmlinuz-[kernel_version]. The second command will install the dynamically loadable kernel modules in a subdirectory of "/lib/modules", named after the kernel version. The resulting modules have the suffix ".ko". For example, if you chose to compile the network device driver for the Realtek 8139 card as a module, there will be a kernel module name 8139too.ko. The third command is OS specific and will copy the new kernel into the directory "/boot".

Next, you may need to create an initrd image in the /boot directory. If you compiled the needed drivers into the kernel then you will not need this ramdisk file to aid in booting. If you need, you can create this ramdisk using the following command:

```
mkinitramfs –o /boot/initrd.img-4.xxx 4.xxx
```

Finally, the following command updates the Grub bootstrap loader configuration file "/boot/grub/grub.cfg" to include a line for the new kernel. The third command, make install, performs many operations behind the scenes. Examine the /etc/grub.d/ directory structure **before** and **after** you run the above commands to see the changes. Also look in the /boot/grub/grub.cfg file for your kernel entry.

```
update-grub
```

If there are error messages from any of the *make* stages, you may be able to solve them by going back and playing with the configuration options. Some options require other options or cannot be used in conjunction with some other options. These dependencies and conflicts may not all be accounted-for in the configuration script. If you run into this sort of problem, you are reduced to guesswork based on the compilation or linkage error messages. For example, if the linker complains about a missing definition of some symbol in some module, you might either turn on an option that seems likely to provide a definition for the missing symbol, or turn off the option that made reference to the symbol.

Reboot the system, selecting your new kernel from the boot loader menu. Watch the messages. See if it works. If it does not, reboot with the old kernel, try to fix what went wrong, and repeat until you have a working new kernel. (If you want to learn more about Grub, you will find that you have options to temporarily modify menu lines on-line, to work around typos in the Grub configuration, but that is beyond the scope of these instructions.)

When you reach a point of frustration, from too many cycles of pruning, recompiling, testing, crashing, and rebooting, stop. Take whatever kernel you have that will compile and run, and go on to the next step. If you are working on a machine in the lab and have not succeeded in producing *any* working kernel from the 4.xxx source tree, get a copy of default configuration file available at /boot, copy it into your Linux source directory under the name ".config", run "make oldconfig", and then "make; make modules_install; make install". It should give you a modest-sized kernel that ran on at least one of the machines in the lab.

3. Material complementar

Compilação do kernel utilizando o makefile do kernel (recomendado):

- http://www.diolinux.com.br/2017/07/como-compilar-um-kernel-linux-passo-a-passo.html
- http://www.thegeekstuff.com/2013/06/compile-linux-kernel/

Compilação do kernel utilizando o utilitário make-kpkg:

- http://helio.loureiro.eng.br/index.php/unix-live-free-or-die/14-linux/304-compilando-kernel-linux-em-debian-e-ubuntu.html
- http://technote.thispage.me/index.php/2016/12/20/ubuntu-16-04-kernel-compile-on-default-setting/

4. Resultado

Esta atividade deve ser acompanhada de um relatório com as seguintes partes obrigatórias:

- Introdução, indicando o que se pretende com o experimento;
- Descrição dos principais passos realizados na montagem do ambiente Linux, informando que configurações foram feitas no ambiente e que serviços foram desabilitados, detalhando alguns dos comando utilizados e demonstrando os resultados obtidos através de imagens;
- Descrição dos principais passos realizados na configuração, compilação, instalação e teste do kernel Linux, detalhando alguns dos comando utilizados e demonstrando os resultados obtidos através de imagens;
- Conclusão indicando o que foi aprendido com o experimento.

Entrega

A entrega da atividade deve ser feita de acordo com o cronograma previamente estabelecido.

Em todos os arquivos entregues deve constar **OBRIGATORIAMENTE** o nome e o RA dos integrantes do grupo.

Devem ser entregues os seguintes itens:

i. Versão final do arquivo de configuração do kernel (.config) utilizado.
ii. Relatório final do trabalho, em formato pdf.

Solicita-se que **NÃO** sejam usados compactadores de arquivos.

Não serão aceitas entregas após a data definida. A não entrega acarreta em nota zero no experimento.