

# CSIS0230A Principle of Operating Systems (Class A)

## Notes for Tutorial 3

### On Linux installation and kernel recompilation

We shall be doing a few simple things in the tutorial this week. Since the things are so simple, the reading material this week is a bit thin. It covers two things: to install and manage a Redhat Linux distribution, and to create, configure and install a new Linux kernel into it.

#### 1. Installing Redhat: an overview

Software installation can be seen as the process of copying some pre-compiled software from an installation media, like a floppy, to a working filesystem in the hard disk, and to configure the system to properly run the software. The installation of an OS like Redhat Linux is no different, except that even the filesystem itself have to be made from scratch. It involves the followings.

1. Boot from an installation media, typically the Redhat CD-ROM. You might need to configure BIOS to boot from the CD-ROM.
2. Set up the keyboard and the mouse.
3. Choose the software packages to install: you can either choose a predefined set of packages or select from the list of all software.
4. Set up the new filesystem. See below.
5. Set up the network, by specifying the network parameters like the IP address.
6. Create the root and user accounts, and set their passwords.
7. Set up the video system for X windows.
8. Wait until all the packages are copied from the installation media to the new filesystem.

#### 2. The filesystem

During the installation of an OS, filesystems are created in the hard disk. Multiple filesystems can all be seen from Linux at the same time, by “mounting” filesystem over different directories. For example, one can create a filesystem for the / directory and another for the /home directory. Then the first filesystem will be used for everything other than the files within /home .

Each filesystem uses one partition of the hard disk, and requires some space. If the whole disk is used for another operating system, (usually Windows 98), then it is necessary to remove or shrink some of the old partitions before making new ones.

One should **create at least two new partitions** for Linux, one for the filesystem, and one for the *swap space*—disk space dedicated for storing memory contents which must be temporarily swapped out of physical memory for other programs to run. Typically, **swap space is made twice the size of physical memory**, which allows a large number of programs to run, without bringing the machine too slow during the maximum usage of swap.

A regular filesystem must host the **Linux kernel**, which resides in the /boot directory. Many older boot loaders, LILO (the one used by Redhat) included, can only boot a kernel installed in the first few giga-bytes of the disk. This forces the kernel to be installed in a relatively small filesystem at the beginning of the disk. In order to use more space, one can use a separate filesystem for /boot , allowing the root filesystem to be as large as desired.

LILO itself is not installed in the filesystem, since it must be installed by a program (Bootstrap of the ROM) that don't understand the Linux (or any other) filesystem. It can be placed in the first few sectors of the first hard disk, typically called the **Master Boot Record** (MBR) of the disk. It can also be placed in the first few sectors of partition hosting the root filesystem, i.e., the *root partition*. For novices, It is probably easier to store use the MBR, although it is easier to remove Linux if you choose to use the root partition instead.

### 3. Redhat package management

A Redhat system is composed of a large number of software sets, called **Redhat Packages**. Each of these packages can be independently installed or removed. A package is distributed as a **RPM** file, with the extension `.rpm`. Most packages have RPMs in two forms: architecture dependent **binary RPM**, and architecture independent **source RPM**. To install a new package, type

```
rpm -i <rpm-file-name>
```

On the other hand, if a different version of the same package is already installed, one would use the following instead to upgrade the package:

```
rpm -U <rpm-file-name>
```

To uninstall a package, one can use

```
rpm -e <package-name>
```

where `<package-name>` is the same as the filename when the package is installed, except that it does not contain the `.rpm` extension.

A binary RPM contains *precompiled programs*, which will be installed in various directories depending on the types of file to install. For example, most programs are installed in the directory `/usr/bin`, while configuration data resides in `/etc`. In contrast, a source RPM contains *source code* to be compiled, which are always installed in the directory `/usr/src/redhat`. The actual source code can be found as a compressed file in `/usr/src/redhat/SOURCES`. To make it easy to compile such packages, all such packages provide a "RPM spec file" in `/usr/src/redhat/SPECS`, which specifies how to extract and compile the source. Compiling using a source RPM package is thus easy:

```
rpm -ba <spec-file-name>
```

builds an RPM in `/usr/src/redhat/RPMS`, which can be installed as described above.

### 4. Compiling your own kernel

It is not uncommon that a custom kernel is needed for a Redhat system. Some reasons for building your own kernels include:

- To **upgrade the kernel**, e.g., to remove a security hole or other bug found in a previous kernel version.
- To **reduce the kernel size**, by removing support of various portions of the kernel. This can dramatically reduce the size of the kernel within memory.
- To apply **special patches** to the kernel to add various capabilities, e.g., to make the kernel preemptive, or to support a journaling filesystem.

- To modify the **configuration** of the kernel, e.g., to allow Linux to use memory over the normal 1G boundary.
- To make your own **kernel modifications**. We will try this in the next week.

In order to build your own kernel, follow these steps.

1. **Download the new kernel.** All kernels released so far can be found in the web site <http://www.kernel.org/>. It is recommended that you only upgrade to a different patch level of the same kernel (e.g., from 2.2.14 to 2.2.19), since even a kernel with different minor number (e.g., 2.4.9) usually need other user programs to be upgraded.
2. The new kernel is in a tar-gzipped file (or tar-bzipped file). Use `tar -xzvf <filename>` (for bzipped files, `tar -xIvf <filename>`, or `tar -xjvf <filename>`, depending on the version of `tar`) to **extract the kernel** to the home directory of root. Move the whole `linux` directory to `/usr/src/linux-MM-mm-pp`, where `MM-mm-pp` is the version number of the Linux you are going to install (e.g., 2.2.19). This directory is called the *kernel root directory*. You should update the symbolic link `/usr/src/linux` to point to this directory, and `cd` to it.
3. It is time to **configure the kernel**, by making a `.config` file in the kernel root directory. If you want, you can **reuse the configuration of Redhat**, which is stored in `../linux-2.2.14/configs/`. Simply choose one of the config files there and copy it to `.config` of the new kernel root directory; and run `make oldconfig`. This alerts you about all the new kernel options, allowing you to either accept it in the main kernel, compile it as kernel module, or not include it.
4. If you want to **modify other kernel options**, you should run `make menuconfig` now. This allows you to decide whether or not to compile each part of the kernel, and whether they should be compiled into the kernel or compiled as a kernel module (to be loaded and unloaded dynamically).
5. Now you are ready to actually **compile the kernel and modules**, by running `make dep` followed by `make zImage`, `make modules`, `make modules_install` and subsequently `make zlilo`.
6. If everything goes well, you have a kernel file `zImage` in the root directory, along with the map file `System.map`; while the module files should already be in `/lib/modules/MM-mm-pp/`. **Move the kernel and map file to the /boot directory** so that they can be used for booting.
7. **Modify the LILO configuration file** `/etc/lilo.conf`, so that it can boot your new kernel. It suffices to make a copy of the last entry of the file, making sensible changes like changing the boot file and the label. Leave the original one there, so that you can boot the old kernel if anything goes wrong. Now **run `lilo` to activate the changes, and reboot.**

One final note: if you choose to reduce the size of the kernel, remember to keep **Second extended fs support** and **Kernel support for ELF binaries** in kernel (not as modules!). Doing otherwise will earn you a completely unbootable kernel.