Linux Usage and Administration Basics

Operating Systems I

Victor Yacovlev (Viktor Iakovlev)

The Boot Stages

- BIOS
- Bootloader GRUB
- Kernel and Initial RAM Drive
- Additional Kernel Modules
- Daemons
- Login Screen

The BIOS

- Basic Input Output System
- Provides the basic functionality required to load an Operating System

The Bootloader

- A small program located on disk at fixed position
- Trivial bootloader just loads The Kernel from known position on disk
- Complex bootloader has access to The Filesystem and can find a file to load
- Complex bootloader is split into two parts: stage 1 (512 bytes) and stage 2 (has no limitation on size)

Kernel and Initial RAM Drive

- The Kernel – is a heart of an Operating System
- Kernel Modules is like “Device Drivers” on Windows
- Some of them are required as early as possible
- There is as initial virtual file system to be loaded by Bootloader

Additional Kernel Modules

- Device drivers and various parts of Kernel functionality
- Might be loaded or not by configuration

Daemons

- Regular programs running at background
- There are system services and real server applications
- Might have dependencies on each other

How Do Daemons Start

- Classic UNIX way (BSD systems):
  a shell script (or set of scripts) which is executed after boot
- Modern Linux way:
  special program called systemd, which starts the first after The Kernel ready

The Login Screen

- The are multiple login sessions allowed
- Login screen is like a daemon running foreground
- Console logins
- Graphical login
- Remote logins (via ssh)

User Session

- Each User has an attribute: a shell program. Usually BASH
- The shell program initializes environment variables from ~/.profile and ~/.bashrc
- Common environment variables for all Users might be specified at /etc/profile and /etc/bashrc

Who is a User

- The God. UID = 0, called ‘root’
- Regular Users. UIDs starts from 1000
- Fake Users used by Daemons. UIDs from 1 to 999

Software Installation

Software Sources

- RPM/DEB files
- Distribution Repositories
- Build from The Source Code
- Custom Installers

Repositories

- Storage of software packages
- Packages have dependencies
- Each distro have a tool to install a package and all its dependencies
- Also used to keep packages up-to-date

Package Installation

Debian/Ubuntu:

- apt-get update   # update packages database
- apt-get install PACKAGE_NAME # install package

Fedora/RedHat/CentOS:

- yum refresh # update packages database
- yum install PACKAGE_NAME # install package

Types of Packages

- Programs
- Libraries used by Programs
- Development Packages for Libraries (header files etc.)
- Metapackages (Virtual Packages) – just have dependencies on other packages to install all of them

Packages Architectures

- Processor-Dependent: i386, x86_64, etc.
- Processor-Independent: noarch

Installation from Sources

- Most programs for Linux are Open Source
- Distributed as archive of source files
- Requires development packages for dependent libraries
- Usually but not always uses common installation method:

```
./configure
make
make install # as root user
```

Installation from Sources

Pros:

- You can install never versions of software than shipped within distribution
- You can create custom configuration or even hack the software before install

Cons:

- The installed source is not managed by package system

Search path

- The root directory / has bin, lib etc
  - This is ‘emergency’ set of files in some distros
- The /usr directory has similar structure
  - The software managed by package system
- The /usr/local directory
  - The software NOT managed by package system, for example built from sources

Search path: PATH=/bin:/usr/bin:/usr/local/bin