Linux init

CSC 510

Linux init

- The `init` process, PID 1, is responsible for ensuring the system runs the correct complement of services and daemons at any given time.
- Common Linux system modes
    - Single-user mode: minimal filesystems mounted, no running services, and a root shell on the console.
    - Multiuser mode: all customary filesystems mounted, all configured network services started, and a GUI window system and login manager.
    - Server mode: similar to multiuser mode, but no GUI systems running.

Some init Setup Tasks

- Set the name of the computer
- Set the time zone
- Check disks with `fsck`
- Mount filesystems
- Remove old files from `/tmp`
- Configure network interfaces
- Start daemons and network services

Implementations of init

- "Traditional" init: based on running tiers of shell scripts to initialize the system
- BSD Unix init: a variant of the traditional init
- `systemd`: a collection of processes for all daemon and state related issues; manages more subsystems than traditional init. (Linux only)

`systemd` Units and Unit Files

- An entity managed by `systemd` is called a unit.
- Example units:
  - a service
  - a socket
  - a device
  - a mount point
  - a startup target
  - a timer
- The behavior of each unit is defined is defined by a unit file; which are located in several places:
  - `/user/lib/systemd/system`
  - `/lib/systemd/system`
  - `/etc/systemd/system`
  - `/run/systemd/system`

Example Unit File

- Example

```
[Unit]
Description=fast remote file copy program daemon
ConditionPathExists=/etc/rsyncd.conf

[Service]
ExecStart=/usr/bin/rsync --daemon --no-detach

[Install]
WantedBy=multi-user.target
```

- Note: details about unit file syntax are in the `systemd.unit` man page

The `systemctl` Command

- `systemctl` is a command for inspect the status of `systemd` and make configuration changes.
- The first argument to `systemctl` is a subcommand
- Examples:
  - `systemctl list-units --type=service`
  - `systemctl list-unit-files --type=service`

Common `systemctl` subcommands

Subcommand	Function
`list-unit-files`	show installed units
`enable` *unit*	enable *unit* to activate at boot
`disable` *unit*	prevent *unit* from activating at boot
`isolate` *target*	changes operating mode to *target*
`start` *unit*	activate *unit* immediately
`stop` *unit*	deactivate *unit* immediately
`restart` *unit*	restart *unit* immediately
`status` *unit*	show *unit*'s status
`kill` *pattern*	send signal to units matching *pattern*
`reboot`	reboot computer
`daemon-reload`	reload unit files and `systemd` configuration

Unit Status

- The `systemctl status` command shows the status of a unit file
- Unit file statuses
  - `bad` – a problem with `systemd`; usually a bad unit file
  - `disabled` – present but not configured to start on boot
  - `enabled` – installed and runnable; will start on boot
  - `indirect` – disabled, but has peers in `Also` clauses that may be enabled
  - `linked` – unit file available through symbolic link
  - `masked` – logically invisible to `systemd`
  - `static` – depended on by another unit

Targets

- A `systemd` target defines a distinct class of units that correspond to common operating modes.
- Some `systemd` targets:
  - `poweroff.target` – system halt
  - `emergency.target` – bare-bones shell for system recovery
  - `rescue.target` – single-user mode
  - `multi-user.target` – multiuser mode
  - `graphical.target` – multiuser mode with GUI
  - `reboot.target` – system reboot

Unit Dependencies

- `systemd` has some implicit dependencies and assumptions (see the systemd.*unit-type* man page) and explicit dependencies declared in the `[Unit]` section of a unit file.
- Explicit dependencies
  - `Wants` – units that should be coactivated if possible, but are not required
  - `Requires` – strict dependencies; failure of any prerequisite terminates this service
  - `Requisite` – strict dependencies that must be active
  - `BindsTo` – similar to `Requires`, but more tightly coupled
  - `PartOf` – similar to `Requires`, but affects only starting and stoppin
  - `Conflicts` – negative dependencies; cannot be coactivated
- A unit's `Wants` or `Requires` can be extended with the `systemctl add-wants` and `systemctl add-requires` subcommands.

`systemd` Timers for Periodic Processes

- A `systemd` timer comprises two files:
  - A timer unit that describes the schedule and unit to activate
  - A service unit that specifies the details of what to run
- Timer types
  - `OnActiveSec` – relative to the time at which the timer itself activated
  - `OnBootSec` – relative to system boot time
  - `OnStartupSec` – relative to the time at which `systemd` started
  - `OnUnitActiveSec` – relative to the time the specified unit was last active
  - `OnUnitInactiveSec` – relative to the time the specified unit was last inactive
  - `OnCalendar` – specific day and time

Aside: cron

- The cron daemon is the traditional tool for running periodic commands.
- The cron configuration file is called a “crontab” for “cron table”; user crontabs are stored at `/var/spool/cron`
- The `crontab` command is used to management crontab files.

Aside: crontab format

- Each non-comment line in a crontab has the following syntax

  ```
  minute hour dom month weekday command
  ```

  - `minute` – minute of the hour; range 0 - 59
  - `hour` – hour of the day; range 0 - 23
  - `dom` – day of month; range 1 - 31
  - `mont` – month of year; range 1 - 12
  - `weekday` – day of week; range 0 - 6 (0 = Sunday)

- The time related fields can contain:
  - A star (*) which matches everything
  - A single integer
  - Two integers separated by a dash, matches a range
  - A range followed by a slash and step value
  - A comma-separated list of integers or ranges