Linux Users Group @ UT DALLAS

The Scary Details of Init

Contact

lug.utdallas.edu

lug.utdallas.edu/chat
irc.oftc.net:6697/#utdlug

utdlug@gmail.com

Officer Positions

The Linux Users Group is looking for students to fill officer roles for Spring 2014!

Requirements:

- Being active in IRC
- Contributing to monthly meetings

Let us know if you're interested!

utdlug@gmail.com

Boot Process Timeline

AC POWER > CMOS > POST > BIOS / EFI > MBR / GPT > BOOT LOADER (GRUB) > INITRAMFS/INITRD + KERNEL > INIT > USERSPACE

Init (short for *initialization*)

- Init is the first process started during booting of the computer system.
- Init is a daemon process that continues running until the system is shut down.
- It is the direct or indirect ancestor of all other processes and automatically adopts all orphaned processes.
- Init is started by the kernel using a hard-coded filename, and if the kernel is unable to start it, a kernel panic will result.
- Init is typically assigned process identifier 1.

-Wikipedia

Linux

SysV - "System 5" style init uses files in /etc/init.d with runlevel configuration in /etc/inittab.

Systemd - Uses .service files in .target directories under /etc/systemd/system. Symlinks to /usr/lib/systemd enable and disable services on boot.

SysV

- http://www.itworld.com/article/2693438/operating-systems/unix-how-to--the-linux--etc-inittab-file.html
- http://www.cyberciti.biz/tips/linux-write-sys-v-init-script-to-start-stop-service.html

SysV Runlevels

- System V init examines the /etc/inittab file for an :initdefault: entry, which defines any default runlevel. If there is no default runlevel, then init dumps the user to a system console for manual entry of a runlevel.
- The runlevels in System V describe certain states of a machine, characterized by the processes run. There are generally eight runlevels, three of which are "standard":
  - 0. Halt
  - 1. Single user mode (aka. *S* or *s*)
  - 6. Reboot
- Aside from these, every Unix and Unix-like system treats runlevels a little differently. The common denominator, the /etc/inittab file, defines what each runlevel does (if they do anything at all) in a given system.

Systemd

- *systemd* uses *targets* which serve a similar purpose as runlevels but act a little different.
- Each *target* is named instead of numbered and is intended to serve a specific purpose with the possibility of having multiple ones active at the same time.
- Some *target*s are implemented by inheriting all of the services of another *target* and adding additional services to it.
- There are *systemd target*s that mimic the common SystemVinit runlevels so you can still switch *target*s using the familiar telinit RUNLEVEL command.

SysV Runlevel	systemd Target	Notes
0	runlevel0.target, poweroff.target	Halt the system.
1, s, single	runlevel1.target, rescue.target	Single user mode.
2, 4	runlevel2.target, runlevel4.target, multi-user.target	User-defined/Site-specific runlevels. By default, identical to 3.
3	runlevel3.target, multi-user.target	Multi-user, non-graphical. Users can usually login via multiple consoles or via the network.
5	runlevel5.target, graphical.target	Multi-user, graphical. Usually has all the services of runlevel 3 plus a graphical login.
6	runlevel6.target, reboot.target	Reboot
emergency	emergency.target	Emergency shell

Creating Custom Unit Files

- http://www.freedesktop.org/software/systemd/man/systemd.service.html
- Dependencies:
  - The most typical case is that the unit *A* requires the unit *B* to be running before *A* is started.
  - In that case add Requires=*B* and After=*B* to the [Unit] section of *A*. If the dependency is optional, add Wants=*B* and After=*B* instead. Note that Wants= and Requires= do not imply After=, meaning that if After= is not specified, the two units will be started in parallel.
  - Dependencies are typically placed on services and not on targets. For example, *network.target* is pulled in by whatever service configures your network interfaces, therefore ordering your custom unit after it is sufficient since *network.target* is started anyway.

Example

```
[Unit]
Description=Mozilla Firefox Sync Server
Requires=network.target
After=network.target

[Service]
Type=forking
ExecStart=/usr/bin/su - moz_sync -c "/usr/bin/screen -d -m -S sync
/home/moz_sync/syncserver/local/bin/pserve /home/moz_sync/syncserver/syncserver.ini"
ExecStop=/usr/bin/su - moz_sync -c "/usr/bin/screen -S sync -X quit"

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

More Examples

https://wiki.archlinux.org/index.php/Systemd/Services

Upstart

- Originally developed by Canonical for Ubuntu
- Full replacement for System V
- Event Based
- Now a dormant project

Upstart - Key Features

- Uses D-Bus for event signaling
- Automatic service respawn
- Slightly clearer specification than System V
- Session based init

Upstart - Example Service File

```
pre-start script
   mkdir -p /var/log/yourcompany/
end script

respawn
respawn limit 15 5

start on runlevel [2345]
stop on runlevel [06]

script
su - youruser -c "NODE_ENV=test exec /var/www/yourcompany/yourproject/yourservice.js 2>&1" >> /var/log/yourcompany/yourservice.log
end script
```

Attribution: https://gist.github.com/c4milo/940909

Upstart - Command Line

- Uses keywords start/stop
  - Ex. “sudo start coffee” to start the coffee service
- Can also run commands of the “service <name> start|stop|restart|reload” variety

BSD

- Every BSD is different.
  - FreeBSD has runlevels (0, 1, 6, c, q) and signals
  - OpenBSD has no runlevels (Uses signals)
- man 8 init / man 8 rc
- init spawns some ttys
- runs /etc/rc which does the rest of the work

BSD: /etc/rc

- source /etc/rc.conf
- turn on swap and fsck
- load default pf rules
- . /etc/netstart
- load user’s pf rules
- mount /usr, /var
- make keys and randomness
- start early/RPC daemons
- mount rest of drives

BSD: /etc/rc

- . /etc/rc.securelevel
- Start network daemons
- . /etc/rc.firsttime
- pkg_scripts
- . /etc/rc.local
- Start local daemons

- Don’t edit /etc/rc use /etc/rc.local
- There’s also /etc/rc.shutdown

BSD: /etc/rc.conf

```
sshd_flags=""       # for normal use: ""
named_flags=NO      # for normal use: ""
nsd_flags=NO        # for normal use: "-
c /var/nsd/etc/nsd.conf"
```

- Override in rc.conf.local
- pkg_scripts
  - `pkg_scripts="dbus_daemon cupsd"`
  - Shutdown stops in reverse order

BSD: rc.d files

- So uniform, all covered under man 8 rc.d
- This is by having all source rc.subr
  - Provides -d and -f for debugging and force
  - Provides check, start, stop, reload, restart
  - Also uses config in /etc/rc.conf
- Writing is covered under man 8 rc.subr
- Can override if needed

BSD: rc.d example

```
#!/bin/sh
daemon="/usr/local/sbin/dhcpd"
. /etc/rc.d/rc.subr
rc_reload=NO
rc_pre() {
        touch /var/db/dhcpd.leases
}
rc_cmd $1
```

BSD: another example

```
#!/bin/sh
#
# $OpenBSD: unbound.rc,v 1.3 2012/08/04 20:43:54 sthen Exp $

daemon="/usr/local/sbin/unbound-control"
daemon_flags="-c /var/unbound/etc/unbound.conf"

. /etc/rc.d/rc.subr

pexp="unbound${daemon_flags:+ ${daemon_flags}}"
rc_reload=NO
```

BSD: another example

```
rc_pre() {
        if ! [[ -f /var/unbound/etc/unbound_server.pem ||
            -f /var/unbound/etc/unbound_control.key ||
            -f /var/unbound/etc/unbound_control.pem ]]; then
                /usr/local/sbin/unbound-control-setup >/dev/null 2>&1
        fi
}
rc_start() {
        ${rcexec} "${daemon} ${daemon_flags} start"
}
rc_check() {
        ${daemon} ${daemon_flags} status
}

rc_stop() {
```