Introduction to Real Life Administration in Linux (part II)

Orr Dunkelman

orrd@vipe.technion.ac.il

Processes

In the UNIX environment, every program which is executed is a process (or can become more than one process).

The system keeps for each process important data, like the process id (pid) and the running environment (memory, priority, nice priority), and a table named file descriptors, which contain information about files (or devices) which the process interacts with.

Processes - Fork

Process can create a son-process using the command fork().

Once a process called for fork, the system creates a new process, with the same (!) parent process' running environment (memory, file descriptor table, etc.).

There is a mechanism to distinguish between the father and the son (or mother and daughter, for the politically correctness), which enables one process to run several sub-processes (sons).

Most common implementation for such a case is a system daemon (like SSH daemon or web-server), that for each new request, copy itself (fork) and one part (father) wait for more requests, and the second one (son) handle with the request.

Processes - Signals and Murderers

Each process has a possibility to get signals. Signals are a way of the system (or the user) to alert a process to a fact.

Signals are used to communicate between processes (the system itself is a pack of processes), and carry information, like the fact that soon the system is going down due to power failure (SIGPWR).

Processes - Signals and Murderers (cont.')

Signal may be caught for the process to handle. For example, we might want the process to be able to de-allocate resources he caught before exiting. However, some signal can not be caught.

If a process is stuck, we can send to it a special signal, called SIGKILL (use only if you are a murderer!). SIGKILL will terminate the process, and free all allocated things the process has.

Be very careful when using SIGKILL, you might kill something important, so use it only when you know what you are doing.

Signals are being sent by user (or administrator) using the command $kill\ \ < signal > \ \ < pid >$.

Processes - Signals and Murderers (cont.')

Common signals include:

- SIGINT - signal interrupt, sent to the process once CTRL Break is pressed.
- SIGKILL - signal kill, sent by the system to end the process. Cannot be caught.
- SIGSTOP - signal stop, the process is frozen. Can be sent from keyboard (press CTRL+Z).
- SIGCONT - signal continue, defrost a frozen process.
- SIGSEGV - signal invalid memory reference. Also known as "Segmentation Fault (celebration)".
- SIGHUP - signal hang-up detection. Now days mostly used to restart programs after their configuration file has been changed.

Processes - Family Bonds

What is connection between father and son?

If the son terminates, the father gets a signal telling him that his beloved son died (SIGCHLD). This is the usual case (in the UNIX model) that the father watches the death of his children. But what if the father dies first?

The assumption is that father is killing his children when he terminates. If that doesn't happen, his children become what is called "zombies". They wait till the father properly terminates, if this happen they'll be terminated by the system.

Processes - Question

If everything is a process, and all processes have a father (who did fork), who is the first process?

Init

Init is the parent of all processes. Its primary role is to create processes from a script stored in the configuration file */etc/inittab*. This file usually has entries which cause init to spawn gettys on each line that users can log in. It also controls autonomous processes required by any particular system.

Run Levels

Run levels are a software configuration of the system which allows only selected group of processes to exist. The processes spawned by init for each of this runlevels are defined in the /etc/inittab file. Init can be in one of eight runlevels 0-6/s/S (where s and S are the same). Runlevel is changed by having a privileged user run the command *telinit*, which activates init.

Runlevels 0,1 and 6 are reserved. Runlevel 0 is used to halt the system. Runlevel 6 is used to reboot the system. Runlevel 1 is used for single user mode. S and s are also a single user mode, but are to be used in scripts etc. instead in real life.

Run Levels (cont.')

Runlevel 7-9 are also valid, but not used (as they are not used in other UNIX systems).

Runlevel 2 is already a useful station, but without NFS service.

Runlevel 3 is a working Linux box, including all the NFS and other network services.

Runlevel 5 is used for X-windows (note that Runlevel 4 is unused).

The inittab File

```
#
id:3:initdefault:

# System initialization.
si::sysinit:/etc/rc.d/rc.sysinit

l0:0:wait:/etc/rc.d/rc 0
l1:1:wait:/etc/rc.d/rc 1
l2:2:wait:/etc/rc.d/rc 2
l3:3:wait:/etc/rc.d/rc 3
l4:4:wait:/etc/rc.d/rc 4
l5:5:wait:/etc/rc.d/rc 5
l6:6:wait:/etc/rc.d/rc 6
```

The inittab File (cont.')

```
# Trap CTRL-ALT-DELETE
ca::ctrlaltdel:/sbin/shutdown -t3 -r now

# UPS failure?
pf::powerfail:/sbin/shutdown -f -h +2
   "Power Failure; System Shutting Down"

# Power returned before shutdown?
pr:12345:powerokwait:/sbin/shutdown -c
   "Power Restored; Shutdown Cancelled"
```

The inittab File (cont.')

```
# Run gettys in standard runlevels
1:2345:respawn:/sbin/mingetty tty1
2:2345:respawn:/sbin/mingetty tty2
3:2345:respawn:/sbin/mingetty tty3
4:2345:respawn:/sbin/mingetty tty4
5:2345:respawn:/sbin/mingetty tty5
6:2345:respawn:/sbin/mingetty tty6

# Run xdm in runlevel 5
# xdm is now a separate service
x:5:respawn:/etc/X11/prefdm -nodaemon
```

The inittab File (cont.')

Each line is in the form of *id:runlevels:action:process*.

id is a unique sequence of 1-4 characters which identify the entry (for backward compitability).

runlevels lists the runlevels for which the specified action should be taken. Several run levels can be written here, no commas or other separators.

action defines the action.

process specify the process to be executed. If this field starts with + than init will not account this process in utmp and wtmp logs.

The inittab File - Valid actions

A valid action can be one of the form:

- respawn - The process is restarted whenever it terminates. For example, getty.
- wait - The process will be started once the runlevel is entered, and init will wait for its termination.
- once - The same like wait, but without waiting.
- boot - The process will be executed during boot. Runlevel is ignored.
- bootwait - Boot + Wait.

The inittab File - Valid actions (cont.')

- ondemand - Used with the special runlevels 'a','b' and 'c'. Executed only when the demanded runlevel is asked. no change in real runlevel is done.
- initdefault - Announce the default runlevel. process field ignored.
- sysinit - The process is executed while the system is initialized. This process will be executed before boot processes (boot & bootwait).
- powerXXX - this is a section of power handling actions.
- ctrlaltdel - this process will be called whenever the CTRL+ALT+DEL sequence is entered at the system console. Usually will be used to reboot or changing to single user mode.

rc.d

Most of the initialize scripts live in peace and harmony in the */etc/rc.d/* directory.

The first is /etc/rc.d/rc.sysinit, which contain all the initializations after the system had loaded (just after the root file system is re-mounted in read-write mode).

rc.d (cont.')

Another important file in this directory is the rc script which executes other runlevel scripts like those in /etc/rc.d/rc3.d/. Whenever a runlevel is entered, rc executes the S* scripts in the corresponding directory. The scripts are called according to the first two digits (S00-S99) in the name. Once a runlevel is being left, the scripts which run are the K* scripts.

Most of the scripts really leave in /etc/rc.d/init.d/ and the "scripts" in the rc?.d directory are symbolic links to the scripts with the right parameter (K* call to the script with the stop command, S* with the start command).

Init Script Example - atd

```
#!/bin/bash
# /etc/rc.d/init.d/atd
# Starts the at daemon
# processname: atd

# Source function library.
. /etc/rc.d/init.d/functions

# check if the at deamon executable exist
test -x /usr/sbin/atd || exit 0
```

Init Script Example - atd (cont.')

```
# See how we were called.
case "$1" in
  start)
        # start command
        # Check if atd is already running
        if [ ! -f /var/lock/subsys/atd ]; then
            echo -n 'Starting at daemon: '
            daemon /usr/sbin/atd
            echo
            touch /var/lock/subsys/atd
        fi
        ;;
```

Init Script Example - atd (cont.')

```
stop)
        #stop
        echo -n 'Stopping at daemon: '
        killproc /usr/sbin/atd
        echo
        rm -f /var/lock/subsys/atd
        ;;
reload|restart)
        $0 stop
        $0 start
        ;;
```

Init Script Example - atd (cont.')

```
  status)
        status /usr/sbin/atd
        ;;
  *)
        echo "Usage: /etc/rc.d/init.d/atd
             {start|stop|restart|reload|status}"
        exit 1
esac

exit 0
```

Related Commands - Processes

- *ps* - Process Status. Gives information about processes.
- *pstree* - Process Tree. Shows the processes' trees.
- *top* - Presents the process information in table.

Related Commands - Signals

- *kill* - Send singal to process.
- *killproc* - Send kill signal to process.
- *alarm* - Set a timer to get an alarm signal.
- *sleep* - Go to sleep for a period of time.

Related Commands - Init

- *shutdown* - Changes the runlevel.
- *halt* - Changes the runlevel to 0.
- *reboot* - Changes the runlevel to 6.
- *telinit* - Changes the runlevel to other level.