

## 05-3 Userspace Initialization – init.d

Chapter 6

## Initialization

- Chapter 5 – Kernel Initialization
- Chapter 6 – Userspace Initialization

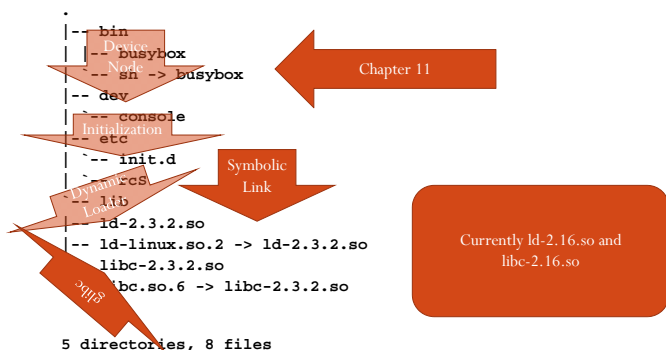
## Chapter 6 - Userspace Initialization

- At startup
  - Kernel initializes
  - Mounts a root file system
  - Executes set of initialization routines
- We'll start with a minimal filesystem and build on it

## Root File System: Top-Level Directories

Directory	Contents
bin	Binary executables, usable by all users on the system
dev	Device nodes (see Chapter 8, "Device Driver Basics")
etc	Local system configuration files
home	User account files
lib	System libraries, such as the standard C library and many others
sbin	Binary executables usually reserved for superuser accounts on the system
usr	A secondary file system hierarchy for application programs, usually read-only
var	Contains variable files, such as system logs and temporary configuration files
tmp	Temporary files

## Minimal File System (Listing 6-1)



## The Embedded Root FS Challenge

- Don't have large hard drive or flash storage
- Hard to tell what depends on what
- Two approaches
  - Trial-and-Error
  - Automated
    - **bitbake** ([www.openembedded.org](http://www.openembedded.org))
    - Buildroot (<http://buildroot.uclibc.org/>)

## Kernel's Last Boot Steps (.../init/main.c)

```
if (execute_command) {
    run_init_process(execute_command);
    printk(KERN_WARNING "Failed to execute %s. Attempting "
        "defaults...\n", execute_command);
}
run_init_process("/sbin/init");
run_init_process("/etc/init");
run_init_process("/bin/init");
run_init_process("/bin/sh");

panic("No init found. Try passing init= option to kernel.");
}

// 2.6.32
```

## Kernel's Last Boot Steps (.../init/main.c)

```
/*
 * We try each of these until one succeeds.
 *
 * The Bourne shell can be used instead of init if we are
 * trying to recover a really broken machine.
 */
if (execute_command) {
    run_init_process(execute_command);
    printk(KERN_WARNING "Failed to execute %s. Attempting "
        "defaults...\n", execute_command);
}
run_init_process("/sbin/init");
run_init_process("/etc/init");
run_init_process("/bin/init");
run_init_process("/bin/sh");

panic("No init found. Try passing init= option to kernel. "
    "See Linux Documentation/init.txt for guidance.");

// 3.2.18 and 3.8.11!
```

## Page 138

- Final sequence of events for the kernel thread called **kernel\_init** spawned by the kernel during the final stages of boot
- **run\_init\_process()** is a small wrapper around the **execve()** function, which is a kernel system call
- **execve()** function *never returns* if no error conditions
- Memory space in which the calling thread is executing from is overwritten by the called program's memory image
- In effect, the called program directly replaces the calling thread, including inheriting its Process ID (PID)

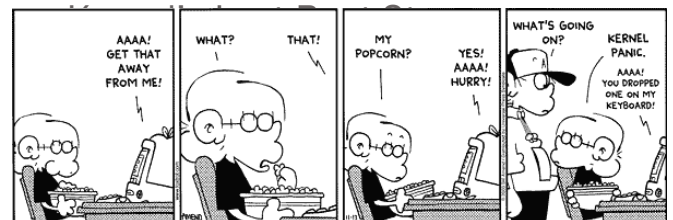
## Kernel's Last Boot Steps

```
if (execute_command) {
    run_init_process(execute_command);
    printk(KERN_WARNING "Failed to execute %s. Attempting "
        "defaults...\n", execute_command);
}
run_init_process("/sbin/init");
run_init_process("/etc/init");
run_init_process("/bin/init");
run_init_process("/bin/sh");

panic("No init found. Try passing init= option to kernel. "
    "See Linux Documentation/init.txt for guidance.");
}
```

## Page 138 (cont.)

- This is the start of user space processing
- Unless the kernel is successful in executing one of these processes, the kernel will halt, displaying the message passed in the **panic()** system call
- If you have been working with embedded systems for any length of time, and especially if you have experience working on root file systems, you are more than familiar with this kernel **panic()** and its message!
- If you search on Google for this **panic()** error message, you will find page after page of hits for this FAQ.
- When you complete this chapter, you will be an expert at troubleshooting this common failure.



```
run_init_process("/bin/sh");
```


```
panic("No init found. Try passing init=
option to kernel.");
}
```

## First User Space Program

- Most systems: `/sbin/init` is spawned.

```
-- bin
|-- busybox
|   '-- sh -> busybox
-- dev
|   '-- console
|   '-- etc
|   '-- init.d
|   '-- rcS
-- lib
|-- ld-2.3.2.so
|-- ld-linux.so.2 -> ld-2.3.2.so
|-- libc-2.3.2.so
|-- libc.so.6 -> libc-2.3.2.so
```

```
run_init_process("/sbin/init");
run_init_process("/etc/init");
run_init_process("/bin/init");
run_init_process("/bin/sh");
```



Busybox is run  
as the initial  
process

## Resolving Dependencies

- You can't put just any program as `init`
- There may be dependencies

```
host$ ldd a.out
linux-gate.so.1 => (0x002df000)
libc.so.6 => /lib/tls/i686/cmov/libc.so.6 (0x00da8000)
/lib/ld-linux.so.2 (0x00a92000)

beagle$ readelf -d a.out | grep NEEDED
0x00000001 (NEEDED)      Shared library: [libc.so.6]
```

## Customized Initial Process

```
console=ttyS0,115200 ip=bootp
root=/dev/nfs init=/sbin/myinit
```

## The `init` process

- Use standard `init`
- Reads `/etc/inittab`

```
# /etc/inittab: init(8) configuration.
# $Id: inittab,v 1.91 2002/01/25 13:35:21 miquels Exp $

# The default runlevel.
id:5:initdefault:

# Boot-time system configuration/initialization script.
# This is run first except when booting in emergency (-b) mode.
si::sysinit:/etc/init.d/rcS
```

## The `init` process

- # What to do in single-user mode.
- `~~:S:wait:/sbin/sulogin`
- # `/etc/init.d` executes the `S` and `K` scripts upon change
- # of runlevel.
- #
- 10:0:wait:/etc/init.d/rc 0
- 11:1:wait:/etc/init.d/rc 1
- 12:2:wait:/etc/init.d/rc 2
- 13:3:wait:/etc/init.d/rc 3
- 14:4:wait:/etc/init.d/rc 4
- 15:5:wait:/etc/init.d/rc 5
- 16:6:wait:/etc/init.d/rc 6

## The `init` process

- # Normally not reached, but fallback in case of emergency.
- `z6:6:respawn:/sbin/sulogin`
- `S:2345:respawn:/sbin/getty 115200 ttyS2`
- # `/sbin/getty` invocations for the runlevels.
- #
- # The "id" field MUST be the same as the last
- # characters of the device (after "tty").
- #
- # Format:
- # `<id>:<runlevels>:<action>:<process>`
- #
- 1:2345:respawn:/sbin/getty 38400 tty1

## Runlevels

Runlevel	Purpose
0	System shutdown (halt)
1	Single-user system configuration for maintenance
2	User defined
3	General purpose multiuser configuration
4	User defined
5	Multiuser with graphical user interface on startup
6	System restart (reboot)

- Runlevel scripts are found in **/etc/rc.d/init.d/**
- or **/etc/init.d/**

## NFS Restart

```
$ /etc/rc.d/init.d/nfs restart
Shutting down NFS mountd: [ OK ]
Shutting down NFS daemon: [ OK ]
Shutting down NFS quotas: [ OK ]
Shutting down NFS services: [ OK ]
Starting NFS services: [ OK ]
Starting NFS quotas: [ OK ]
Starting NFS daemon: [ OK ]
Starting NFS mountd: [ OK ]
```

## Runlevel Directory Structure on 3.2 Beagle

```
beagle$ ls -dl /etc/rc*
drwxr-xr-x 2 root root 4096 Mar 13 20:18 /etc/rc0.d
drwxr-xr-x 2 root root 4096 Mar 13 20:18 /etc/rc1.d
drwxr-xr-x 2 root root 4096 Mar 13 20:18 /etc/rc2.d
drwxr-xr-x 2 root root 4096 Mar 13 20:18 /etc/rc3.d
drwxr-xr-x 2 root root 4096 Mar 13 20:18 /etc/rc4.d
drwxr-xr-x 2 root root 4096 Mar 13 20:18 /etc/rc5.d
drwxr-xr-x 2 root root 4096 Mar 13 20:18 /etc/rc6.d
drwxr-xr-x 2 root root 4096 Mar 13 20:18 /etc/rcS.d
```

## Example Runlevel Directory on 3.2 Beagle

```
beagle$ ls -ls rc5.d/
total 0
0 lrwxrwxrwx 1 root root 20 Mar 13 20:18 S05led-config -> ../init.d/led-config
0 lrwxrwxrwx 1 root root 18 Mar 13 20:18 S10dropbear -> ../init.d/dropbear
0 lrwxrwxrwx 1 root root 14 Mar 13 20:18 S20apmd -> ../init.d/apmd
0 lrwxrwxrwx 1 root root 16 Mar 13 20:18 S20dbus-1 -> ../init.d/dbus-1
0 lrwxrwxrwx 1 root root 16 Mar 13 20:18 S20syslog -> ../init.d/syslog
0 lrwxrwxrwx 1 root root 22 Mar 13 20:18 S21avahi-daemon -> ../init.d/avahi-daen
0 lrwxrwxrwx 1 root root 17 Mar 13 20:18 S22conmman -> ../init.d/conmman
0 lrwxrwxrwx 1 root root 17 Mar 13 20:18 S30ntpdate -> ../init.d/ntpdate
0 lrwxrwxrwx 1 root root 20 Mar 13 20:18 S50usb-gadget -> ../init.d/usb-gadget
0 lrwxrwxrwx 1 root root 16 Mar 13 20:18 S99gpe-dm -> ../init.d/gpe-dm
0 lrwxrwxrwx 1 root root 19 Mar 13 20:18 S99rmnlogin -> ../init.d/rmnlogin
0 lrwxrwxrwx 1 root root 20 Mar 4 22:09 S99zapsplash -> ../init.d/zapsplash
```

## Runlevel 5

```
beagle$ ls /etc/rc5.d | cat
K36cups          INIT: Entering runlevel: 5
S02dbus-1       Starting system message bus: dbus.
S05led-config    Starting Hardware abstraction layer hald
S10dropbear      Configuring leds:
                  beagleboard::pmu_stat: none
S20apmd          beagleboard::usr0: heartbeat
                  beagleboard::usr1: mmc0
                  Starting Dropbear SSH server: dropbear.
                  Starting advanced power management
                  daemon: No APM support in kernel
                  (failed.)
```

## Runlevel 5

```
S20cron          Starting Vixie-cron.
S20samba         Starting Samba: smbd nmbd.
S20syslog        Starting syslog-ng:.
S20xinetd        Starting internet superserver:
                  xinetd.
S21avahi-daemon  * Starting Avahi mDNS/DNS-SD
S28NetworkManager Daemon: avahi-daemon
                  [ ok ]
S30pvr-init      Starting Network connection
S50system-tools-backends manager daemon: NetworkManager.
S50usb-gadget    Starting PVR
S81cups          cups: started scheduler.
S99gdm           Starting GNOME Display Manager
S99rmnlogin      gdm
```

## Beagle 3.8

```
beagle$ cat /etc/init.d/README
```

You are running a systemd-based OS where traditional init scripts have been replaced by native systemd services files. Service files provide very similar functionality to init scripts. To make use of service files simply invoke "systemctl", which will output a list of all currently running services (and other units). Use "systemctl list-unit-files" to get a listing of all known unit files, including stopped, disabled and masked ones. Use "systemctl start foobar.service" and "systemctl stop foobar.service" to start or stop a service, respectively. For further details, please refer to `systemctl(1)`.

## Beagle 3.8 (cont)

```
beagle$ cat /etc/init.d/README
```

Note that traditional init scripts continue to function on a systemd system. An init script `/etc/init.d/foobar` is implicitly mapped into a service unit `foobar.service` during system initialization.

Thank you!

Further reading:

`man:systemctl(1)`

`man:systemd(1)`

<http://0pointer.de/blog/projects/systemd-for-admins-3.html>

<http://www.freedesktop.org/wiki/Software/systemd/Incompatibilities>