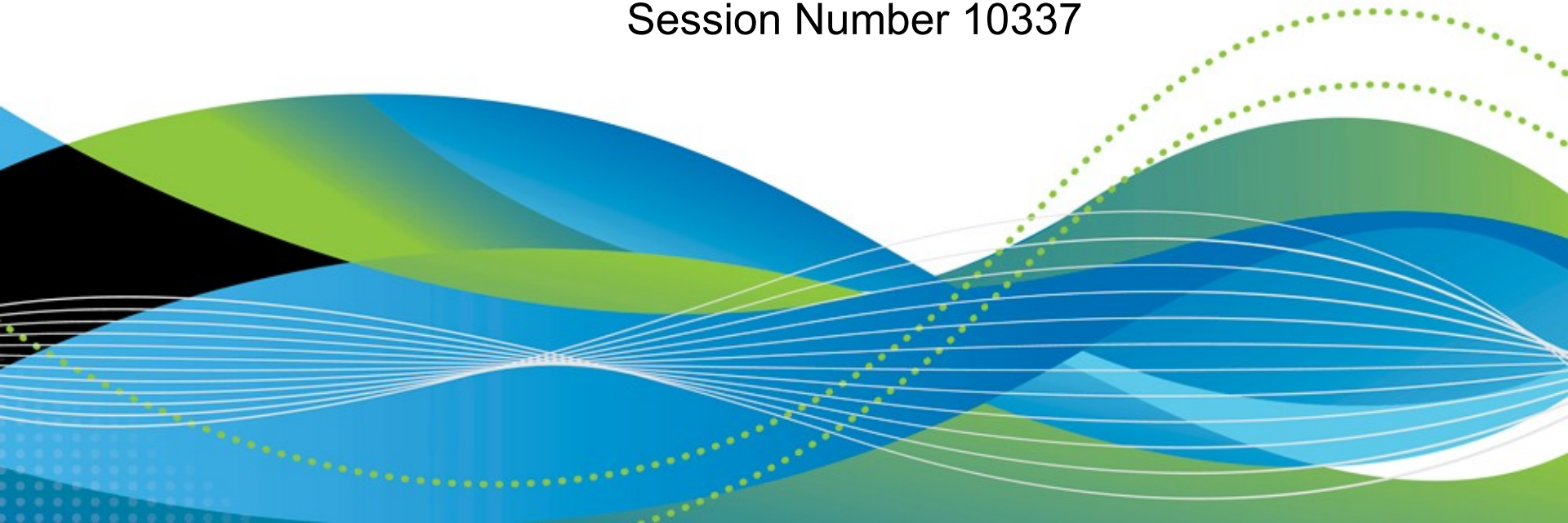
#SHAREorg

SHARE
Technology · Connections · Results

Saving Your Bacon Recovering From Common Linux Startup Failures

Mark Post
SUSE

Tuesday, March 13, 2012
Session Number 10337



Agenda

- How the boot process is *supposed* to work
- What things can go wrong
- How you can get things running quickly
- Review of some recovery scenarios
- How to fix things so the next IPL will work
- How to prevent similar issues in the first place
- Questions

- I will take questions during the presentation, unless time gets short.

How the boot process is *supposed* to work

How the boot process is *supposed* to work

- Hardware (or z/VM) reads the bootstrap record from disk
- The bootstrap record has entries in it pointing to
  - The kernel plus the kernel parameters
  - The initial ram disk (initrd)
- The kernel is read into memory and control is passed to its entry point
  - Soon, like other architectures, it will be compressed on disk, so the kernel will decompress itself first.
- The kernel parameters are passed to the kernel by the boot loader. They get stuffed into a pre-determined address within the kernel itself.

How the boot process is *supposed* to work (2)

- The kernel goes through it's initialization process, which includes parsing the kernel parameters
  - Parameters the kernel doesn't recognize as being for itself are ignored. **All** are put in /proc/cmdline.
- The initrd is read in, decompressed and unpacked
- The file system from the initrd is mounted as the (temporary) root file system
- The kernel finds the userspace program in the initrd to handle the rest of the boot process
  - By default this is “init” but kernel parameters can specify something else, e.g., /bin/bash

How the boot process is *supposed* to work (3)

- The init program begins execution. In SLES, this is a bash script.
- The init program does all sorts of things. The ones we're interested here are:
  - Parsing parameters of interest in /proc/cmdline
    - Some of these are for kernel modules to be loaded, others are not directly related to the kernel or its modules.
  - Loading kernel modules for various hardware with appropriate parms
  - Finding and mounting the permanent root file system
  - Starting execution of the “real” init program

What things can go wrong

What could possibly go wrong?

Ooh, where do we start?

- The boot loader can't find both the kernel and the initrd
- When the kernel tries to uncompress and mount the initrd, it's not valid
- The initrd doesn't contain all the kernel modules you need to activate all your DASD and your NIC
- Your root file system is on an LVM logical volume, but the initrd doesn't have that support built in
- Your root file system is *not* on an LVM logical volume, but the initrd *does* have that support built in

What could possibly go wrong? (2)

- The scripts in the initrd don't activate all the DASD volumes you need
- Your root file system is on a network server, but the initrd can't initialize your NIC
- You've edited /etc/fstab and now there's a typo in it
- You don't remember the root password
- You get a message that the init program can't be found
- None of your kernel modules can be found
- You've told the kernel to ignore the address of the console.

How you can get things running quickly

We're Baaaaack

- If you're going to edit a file, make a backup first.
  - If you don't like that idea, you're going to *hate* learning ed or sed
- Have a small “rescue system” available that you can use to access the DASD or SCSI devices of a system with problems.
- Use the kernel and initrd from the installation media to boot from the z/VM card reader, or the HMC in the case of an LPAR.
  - Activate the DASD needed for the root file system and mount it
  - for fs in dev sys proc; do mount –bind /$fs /mnt/$fs; done
  - mount –bind /proc/mounts /mnt/etc/mtab
  - chroot /mnt

- Understand at what point in the boot process various resources get activated
  - E.g., Network interface initialization happens *after* the initrd is replaced by the real root file system
  - DASD or SCSI device(s) needed by the root file system are brought online in the initrd
- Know how to dynamically activate and bring online
  - DASD
  - LVM volume groups and logical volumes
  - QETH devices (NICs for the most part, but can be FCP adapters)
  - SAN WWPNs and LUNs

- Set up the Terminal Server guest, and configure all the “target” systems to be able to receive those connections.
  - Requires a current enough SLES10 or SLES11 SP1 system
  - Keep in mind that if your root file system can't get mounted, the target system won't be able to receive connections because the code for that lives in the root file system.

How I normally set up my file systems

```
# df -h
Filesystem                Size  Used Avail Use% Mounted
/dev/dasda1               388M  246M  123M  67% /
/dev/mapper/vg1-home       97M  4.2M   88M   5% /home
/dev/mapper/vg1-opt        74M  4.1M   66M   6% /opt
/dev/mapper/vg1-tmp       291M   43M  234M  16% /tmp
/dev/mapper/vg1-usr       2.6G  1.8G  678M  73% /usr
/dev/mapper/vg1-var       392M  298M   78M  80% /var
```

- The mkinitrd is *supposed* to figure out what device type the root file system is on (including network-based), and include the necessary pieces for that in the initrd.
  - Sometimes there are bugs that prevent this from working right.
- On my SLES11 SP1 system, the only kernel modules that in the initrd are:
  dasd_eckd_mod.ko
  dasd_mod.ko
  ext3.ko
  jbd.ko
  mbcache.ko

Commands available in the initrd

bin:

awk	echo	ln	mv	sleep	usleep
bash	grep	logger	rm	test	warpclock
cat	ipconfig	ls	run-init	touch	
chmod	ipconfig.sh	mkdir	sed	true	
cp	kill	mknod	sh	umount	
date	linuxrc	mount	showconsole	uname	

sbin:

blogd	fsck	insmod	reboot	udevd
dasd_configure	fsck.ext3	killall5	resume	
dasdinfo	halt	modprobe	showconsole	
dasdview	ifup	pidof	udevadm	

```
boot:
01-devfunctions.sh
02-start.sh
03-storage.sh
04-udev.sh
05-blogd.sh
05-clock.sh
11-block.sh
11-dasd.sh
21-devinit_done.sh
81-resume.userspace.sh
82-resume.kernel.sh
83-mount.sh
84-remount.sh
91-createfb.sh
91-killblogd.sh
91-killudev.sh
91-shell.sh
92-killblogd2.sh
93-boot.sh
```

Review of some recovery scenarios

How to fix things so the next IPL will work

- Use YaST to define the resource, or
- Use the command line functions provided
  - dasd_configure
  - qeth_configure
  - zfcp_host_configure
  - zfcp_disk_configure
- Either way, the proper udev rules will be written into the /etc/udev/rules.d/ files.
- If in doubt, re-run mkinitrd and zipl.
  - Check to make sure the modules necessary to get the root file system mounted are included.

How to prevent similar issues in the first place

Prevention, Worth a Pound ...

- Use YaST
  - Yeah, I know, that can be a pain at times. But it usually causes fewer problems than humans.
    - **It's also partly why a lot of YaST functions can be called from scripts.**
- After updating /etc/fstab, do a “mount -a” command and look for any errors.
- Really know and understand what *is* in your initrd, and what is *supposed* to be there.
  - Unpack the initrd into a temporary directory and poke around
    mkdir tempdir
    cd tempdir
    zcat /boot/initrd | cpio -ivmd

Prevention (2)

- Understand that what goes into the initrd is only those things needed to get the root file system mounted.
  - Anything beyond that is unnecessary, and in some cases can *cause* startup failures.
- Don't assume that the way things were done in a prior release will be done the same way now.
  - Read the release notes. For every service pack. Yes, I mean it. No, I'm not kidding.
  - zipl.conf updates not needed for DASD adds/deletes
  - /etc/sysconfig/hardware versus /etc/udev/rules.d

Questions