Reducing Startup Time in Embedded Linux Systems

Tim Bird
Chair – Bootup Times Working Group
CE Linux Forum

Overview

- Characterization of the problem space
- Current reduction techniques
- Work in progress
- CE Linux Forum

Characterizing the Problem Space

The Problem

- Linux doesn’t boot very fast
  - Current Linux desktop systems take about 90-120 seconds to boot
- This is clearly not suitable for consumer electronics products

Delay Taxonomy

- Major delay areas in startup:
  - Firmware
  - Kernel/driver initialization
  - User space initialization
  - Application startup
- Scope of problem
  - Device-specific
  - Systemic

Overview of delays

Startup Area	Delay
Firmware	15 seconds
Kernel/driver initialization	9 seconds
RC scripts	35 seconds
X initialization	9 seconds
Graphical Environment start	45 seconds
Total:	113 seconds

For laptop with Pentium III at 600 MHZ

Firmware delays

- X86 firmware (BIOS) is notorious for superfluous delays (memory checking, hardware probing, etc.)
  - Many of these operations are duplicated by the kernel when it starts
- Large delay for spinup of hard drive
- Delay to load and decompress kernel

Typical HD Time-to-Ready

Brand	Size	Time to Ready
Maxtor	3.5”	7.5 seconds
Seagate	3.5”	6.5 - 10 seconds *
Hitachi	3.5”	6 - 10 seconds *
Hitachi	2.5”	4 - 5 seconds
Toshiba	2.5”	4 seconds
Hitachi microdrive	1.0”	1 - 1.5 seconds

* Depends on number of platters

During retries, these times can be extended by tens of seconds, but this is rare.

Load and decompress times

- Typically the kernel is stored in compressed form (zImage or bzImage)
- Entire kernel must be loaded from storage (HD or flash) and decompressed
  - If on HD, there are seek and read latencies
  - If on flash, there are read latencies
- For a slow processor, this can take 1 to 2 seconds

Kernel/Driver startup delays

- Delay calibration
- Probing for non-existent hardware
- Probing PCI bus
- Probing IDE slots
- Probing USB chain
- Driver init is serialized
  - Busy-waits in drivers
- Serial console output

RC scripts

- Set of shell scripts to initialize a variety of user-space daemons and services
- Invoked in sequence
- Time is heavily dependent on the set of services to be initialized
- Overhead for:
  - Interpreting the scripts
  - Loading and executing applications (some applications are loaded multiple times)

Application start

- Time to load shared libraries
- Time to initialize graphics and windowing systems
- Time for applications to load and initialize

Current Reduction Techniques

Primary observation…

Mantra:

Configuration of hardware and software is <u>much</u> more fixed for embedded systems than for desktop systems

Speedup Methods

- Do it faster
- Do it in parallel
- Do it later
- Don't do it at all

Overview of Reduction Techniques

- Execute-in-place (XIP)
- Probe/calibration elimination
  - Elimination of runtime determination of fixed values
- De-serialization
  - Concurrent driver initialization
  - Parallel RC scripts
- Deferring of operations
  - Late driver load
  - Late FS journal log replay

Reduction Techniques for Firmware

Kernel XIP

- Place kernel uncompressed in flash or ROM
- Map or set kernel text segments directly in machine address space
- Details:
  - Use Linear CramFS for kernel
  - Bootloader sets up mapping and transfers control directly to kernel

Kernel XIP pros and cons

- Pros:
  - faster bootup – eliminates load and decompress times for kernel
  - smaller RAM footprint – kernel text segment is not loaded into RAM
- Cons:
  - Adds overhead for running kernel
    - Access to Flash is slower than access to RAM

Kernel XIP results

Boot time for PowerPC 405LP at 266MHZ.

Boot Stage	Non-XIP time	XIP time
Copy Kernel to RAM	85 msec	12 msec *
Decompress kernel	453 msec	0 msec
Kernel time to initialize (time to first userspace prog)	819 msec	882 msecs
Total kernel boot time	1357 msecs	894 msecs

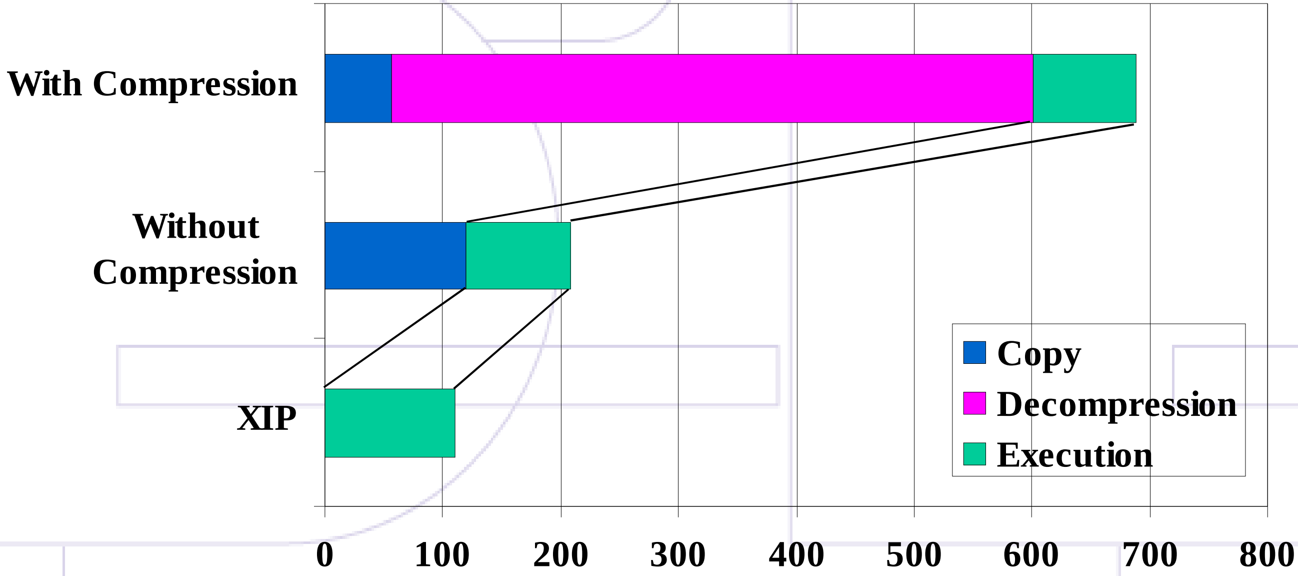
* Data segment must still be copied to RAM

Kernel XIP runtime overhead

Operation	Non-XIP	XIP
stat() syscall	22.4 µsec	25.6 µsec
fork a process	4718 µsec	7106 µsec
context switching for 16 processes and 64k data size	932 µsec	1109 µsec
pipe communication	248 µsec	548 µsec

Results from lmbench benchmark on OMAP (ARM9 168 MHZ)

Kernel XIP Results



	With Compression	W/O Compression	XIP
Copy	56 msec	120 msec	0 msec
Decompression	545 msec	0 msec	0 msec
Kernel execution	88 msec	88 msec	110 msec
Total:	689 msec	208 msec	110 msec

HD Spinup in Parallel with Kernel Init

- Hard drive spinup is one of the most costly operations during startup.
- Can start HD in firmware prior to kernel load
- Obviously, kernel can't reside on HD
  - Requires separate storage for kernel (and possibly other init programs)

Reduction Techniques for Kernel

Pre-set loops_per_jiffy

- Very easy to do:
  - – Measure once (value is BogoMips * 5000)
  - – Set value in init/main.c:calibrate_delay_loop()
  - – Don't perform calibration
- Saves about 250 msec

Don’t probe certain IDE devices

- Can turn off IDE probe with kernel command line:
  - ide<x>=noprobe
  - Requires a bugfix patch (feature was broken in 2.4.20)
- Can also turn off slave devices:
  - eg. hd<x>=none
- Time to probe for an empty second IDE interface was measured at 1.3 seconds

Use Deferred and Concurrent Driver Initialization

- Change drivers to modules
  - Statically compiled drivers are loaded sequentially, with "big kernel lock" held
- Replace driver busywaits with yields
- Load drivers later in boot sequence
  - In parallel with other drivers or applications
- Benefit is highly driver-specific
  - e.g. PCI sound card had 1.5 seconds of busywait
- Requires per-driver code changes

Turn off serial console output

- Probably turned off in final product configuration, but…
- During development, overhead of serial console output (printk output) is high
- Use "quiet" on kernel command line
- Can still read messages from printk buffer after startup (use dmesg)

Reduction Techniques for User Space

Defer replay of FS log

- Ext3 and XFS both replay their log at boot/mount time
- Can mount FS readonly on boot
  - Later, switch to read/write and replay the log to ensure FS integrity
- Requires file system areas to be organized to support deferral of access to writable areas.
  - Put writable areas (e.g. /var) in RAM disk temporarily
- About 200 ms improvement in some tests

Eliminate unneeded RC scripts

Default Script List

anacron.sh
bootmisc.sh
checkfs.sh
checkroot.sh
console-screen.sh
cron.sh
devfsd.sh
devpts.sh
devshm.sh
hostname.sh
hwclock.sh
ifupdown.sh
keymap.sh
modutils.sh
mountall.sh
networking.sh
procps.sh
rmnologin.sh
syslog.sh
urandom.sh

Reduced Script List

bootmisc.sh
checkfs.sh
checkroot.sh
hwclock.sh
modutils.sh
mountall.sh
networking.sh
urandom.sh

Replace RC Scripts with Custom init Program

- Replace scripts and /sbin/init program itself
- Use compiled program instead of shell script
  - Avoids shell invocation and parsing overhead
- Drawbacks:
  - You have to maintain your custom init program
  - System is no longer reconfigurable via file operations

Application XIP

- Requires linear file system (like CramFS or ROMFS)
- Map libraries and applications into address space directly from Flash/ROM
- Good application load performance (on first load)
- Slight performance degradation

Application XIP Results

Time to run shell script which starts TinyX X server and *xsetroot -solid red*, then shuts down

Invocation	Non-XIP	XIP
First time	3195 msec	2035 msec
Second time	1744 msec	1765 msec

System-wide improvements

- Reduce kernel, library and application size by using smallest configuration possible.
  - Reduces load time and can improve cache hits
- Keep read-only and executable data separate from writable data in flash storage
  - Write times (which are long) don't interfere with read times
- Use Linear CramFS for read-only data
  - CramFS has little meta-data and mounts quickly

System-wide improvements (cont.)

- Keep writable files in RAM disk, and migrate to flash after boot
- Reduce the amount of filesystem I/O (especially writes to flash)
- Turn off klogd/syslogd logging to stable storage
- Set library search paths to reduce failed open attempts

Work in Progress

WIP Overview

- Continuing project with Matsushita and MontaVista
- Reduction in RC script overhead
- More probe elimination
- Quick and safe shutdown

Reduction in RC script overhead

- Use of busybox for shell interpreter (ash) and builtin commands
  - Eliminates overhead of large program invocations
- Modification to RC scripts to avoid loading shell multiple times
- Modification to busybox to avoid fork and exec on shell invocations

Reduction in RC script Overhead Early Results

- Time to run set of RC scripts reduced from 8 seconds to 5 seconds
  - On ARM9, 168 MHZ

WIP Availability

- Final results and patches will be available early next year.

Ideas for Future Research

- Pre-linking
  - Pre-calculate relocations and fixups for dynamic libraries
  - KDE and Qt/Embedded use forms of this now
- RC script command results caching
  - Maybe can replace RC script use of find and grep with cached results
- Driver configuration cache
  - Form of hibernate/unhibernate for drivers and bus code

Instrumentation

- Instrumented printk
  - Patch is available now (contact me)
- Can use Kernel Function Instrumentation (KFI) for kernel time measurements
  - MontaVista products include this
- For user space, can use:
  - strace –tt
  - time
  - Linux Trace Toolkit

Remember!

- Do it faster
- Do it in parallel
- Do it later
- Don’t do it at all

Good Luck!