Reducing Boot Time: Techniques for Fast Booting

Chris Hallinan
Field Applications Engineer
Author: Embedded Linux Primer
Agenda

→ Introduction and Overview
  → Hardware Discussion
→ Software Components
  → Overview of Boot Sequence
→ Bootloader Issues
→ Linux Kernel Optimizations
→ Tools
→ Applications/Userland
Introduction

- Fast Boot is Important to Many Products
  - Consumer, Automotive, Medical Devices, etc.

- Significant gains w. minimal investment

- First, define “boot”. Does it mean:
  - Splash screen?
  - 1st user process started?
  - Device fully up and running, connected?

- Boot time is affected by many factors:
  - Hardware Design
  - Bootloader Implementation
  - Kernel Configuration
  - Application Profile
Typical Boot Sequence

- Power Applied
- Pwr/Clk/Reset
- Bootloader
- Kernel Init
- Root FS Mount
- Userland Apps

Optimization Opportunities

- **Bootloader**
  - May be multiple stages
  - Remove support for unused features
  - Modify/remove hardware probing features
  - Keep it Simple, Small :)

- **Kernel**
  - Many opportunities for optimization
  - Low hanging fruit can be easy to 'pluck'

- **Applications**
  - Most are up to you!

- Most initialization is serial
- Take measurements to decide where to focus!
Bootloader has two primary responsibilities:

- Initialize CPU/Hardware (minimally)
- Locate, load and execute a kernel image

- May involve several steps, including device I/O, decompression, etc.
Most bootloaders have many more features

- Esp. look for any type of bus enumeration
- Lots of useful “development” functionality
  - dhcp, tftp, pci scan, mem utils
  - device initialization, Flash utilities, etc
- In a production system, many of these features may be unnecessary
  - Disabling these features can have a significant impact on boot time
- For fast boot, get the bootloader out of the way quickly

Remember, small === fast
Kernel Image Optimization

- Use Uncompressed Kernel
  - Decompression can take several seconds!
  - Tradeoff: more Flash storage required

- Kernel build produces two images*
  - Image and zImage (ARM)
  - Obviously, zImage is the compressed version

- On i.MX31, this saved on average ~750 mS

- YMMV, depending on CPU speed, FLASH speed...

*Details vary for each architecture, ARM discussed here
Eliminate Unnecessary Kernel Options

- Reduces kernel size
- Speeds up kernel loading
- In some cases, will reduce kernel init time once kernel is loaded

Typical default kernel config contains lots of 'stuff' (i.e. features) you may not need:

- MD/Raid support, IPv6, Numerous File Systems, Extended Partition support, etc.
- Debug features such as kernel symbols, ikconfig, etc.
- Many are compiled in features and increase kernel size
Kernel Config Options to Consider Disabling

- **CONFIG_MD**
  - RAID/LVM support

- **CONFIG_IDE**
  - Saves init time if not used on HW w/ IDE ctrlr
  - Can also use hdx=noprobe

- **CONFIG_PCCARD**

- **CONFIG_HOTPLUG**
  - Remove support for hotplug if not required

- **CONFIG_IKCONFIG**
  - Removes support for config info, makes kernel smaller

- **CONFIG_DEBUG_KERNEL**
  - **CONFIG_KALLSYMS**
  - **CONFIG_BUG**
    - Used for debug – can be removed if desired
More Interesting Kernel Config Options

→ Check Networking options
  → Lots of functionality – do you need it all?
  → ie. kernel autoconf, multicast, advanced router, etc.

→ Remove support for unnecessary FS features
  → Default configs often have much of this enabled
    → CONFIG_DNOTIFY
    → CONFIG_INOTIFY
    → CONFIG_XFS
    → CONFIG_AUTOFS4_FS (Automounter)
    → etc

→ Remember: Smaller kernel = Faster load

*If in doubt, make small changes. Test early and often so you can rollback breakages easily!*
Calibration Routines

- Many hardware platforms spend considerable time in calibration routines
  - “Calculating BogoMips…”
  - Allows precise µdelay() routines
  - Can take significant time

- Use kernel command line: loops-per-jiffy:
  - \( \text{lpj} = \text{xxxxx} \)

- Easy to use: most platforms will display correct value in kernel log (and to console) on startup
Consider your system requirements:

- What functionality must be available immediately?
- What functionality can be deferred?
- What drivers are not needed?

Modules (CONFIG_*=M), if unused, are irrelevant
- Won't affect kernel startup time
Drivers can be pre-compiled into kernel or built as modules for loading later

- Use statically-linked drivers for functions that must be immediately available
- Use Loadable Modules for deferred functionality

Caveat: if you can avoid `CONFIG_MODULES`, kernel will be smaller*, thus faster to load. There is probably also a minor hit to reading the driver from the FS instead of it being in the kernel

*Assumes minimal system, few drivers.
File System Selection

- Consider CRAMFS for initial read-only FS
  - Compact and fast
  - No journaling entries to scan on initial mount
- Use tmpfs for /tmp, possibly /var, others
- Mount writable FS later, such as JFFS2 on NOR Flash
- Consider your tolerance to sudden power off
  - Journaling file systems can protect but at a cost of increased startup times
XIP – Execute in Place

- Processor does not copy Kernel image to DRAM
  - Executes directly from (NOR) Flash

- Advantages
  - Reduces DRAM requirements (and thus power)
  - Eliminates time-consuming copy from Flash

- Disadvantages
  - Depending on h/w architecture, could be much slower
    - i.e. burst/cache performance, etc.
  - Cost of Flash – kernel must be stored uncompressed

- Your Mileage May Vary
Remove Support for printk()

- The “Brute Force” approach - CONFIG_PRINTK
  - Completely eliminates calls to printk()

- Advantages
  - Saves significant kernel size, and therefore load time
  - Eliminates many boot messages - decreasing boot time

- Disadvantage
  - No kernel status message are available!
  - Makes kernel debugging very difficult

- A thoroughly tested kernel should work well here

- Alternatively, use “quiet” command line parameter
  - Suppresses printk output during boot, preserving the printk infrastructure to be used post-boot
Tools

- **KFT: Kernel Function Timing**
  - Requires KALLSYMS, mentioned above
  - Provides function call tracing and timing

<table>
<thead>
<tr>
<th>Entry</th>
<th>Duration</th>
<th>Local</th>
<th>Pid</th>
<th>Trace</th>
</tr>
</thead>
<tbody>
<tr>
<td>162</td>
<td>11523</td>
<td>0</td>
<td>0</td>
<td>paging_init</td>
</tr>
<tr>
<td>162</td>
<td>11523</td>
<td>0</td>
<td>0</td>
<td>free_area_init_nodes</td>
</tr>
<tr>
<td>162</td>
<td>11523</td>
<td>12</td>
<td>0</td>
<td>free_area_init_node</td>
</tr>
<tr>
<td>162</td>
<td>11511</td>
<td>11511</td>
<td>0</td>
<td>__etext+0x2f0</td>
</tr>
<tr>
<td>162</td>
<td>11511</td>
<td>0</td>
<td>0</td>
<td>__alloc_bootmem_node</td>
</tr>
<tr>
<td>162</td>
<td>11511</td>
<td>11511</td>
<td>0</td>
<td>__alloc_bootmem_core</td>
</tr>
<tr>
<td>11787</td>
<td>2307</td>
<td>2307</td>
<td>0</td>
<td>vfs_caches_init_early</td>
</tr>
<tr>
<td>11787</td>
<td>1531</td>
<td>69</td>
<td>0</td>
<td>vfs_caches_init_early</td>
</tr>
<tr>
<td>11787</td>
<td>686</td>
<td>0</td>
<td>0</td>
<td>alloc_large_system_hash</td>
</tr>
<tr>
<td>11787</td>
<td>686</td>
<td>0</td>
<td>0</td>
<td>alloc_bootmem</td>
</tr>
<tr>
<td>11787</td>
<td>686</td>
<td>0</td>
<td>0</td>
<td>alloc_bootmem_nopanic</td>
</tr>
<tr>
<td>11787</td>
<td>686</td>
<td>686</td>
<td>0</td>
<td>alloc_bootmem_core</td>
</tr>
<tr>
<td>13318</td>
<td>776</td>
<td>776</td>
<td>0</td>
<td>inode_init_early</td>
</tr>
<tr>
<td>14094</td>
<td>1208</td>
<td>641</td>
<td>0</td>
<td>mem_init</td>
</tr>
<tr>
<td>14094</td>
<td>567</td>
<td>4</td>
<td>0</td>
<td>free_all_bootmem</td>
</tr>
<tr>
<td>14094</td>
<td>563</td>
<td>563</td>
<td>0</td>
<td>free_all_bootmem_core</td>
</tr>
<tr>
<td>15607</td>
<td>3851</td>
<td>3851</td>
<td>0</td>
<td>schedule</td>
</tr>
<tr>
<td>15607</td>
<td>3848</td>
<td>3848</td>
<td>0</td>
<td>schedule</td>
</tr>
<tr>
<td>15630</td>
<td>1573099</td>
<td>1573099</td>
<td>1</td>
<td>kernel_init</td>
</tr>
<tr>
<td>15666</td>
<td>81152</td>
<td>81152</td>
<td>4</td>
<td>schedule</td>
</tr>
<tr>
<td>15666</td>
<td>81139</td>
<td>81139</td>
<td>4</td>
<td>schedule</td>
</tr>
</tbody>
</table>
Tools

Linux Trace Toolkit
Tools

➔ printk timestamps (CONFIG_PRINTK_TIME)
➔ Appends time info to printk() output
➔ Enables measurement of long operations, esp. at boot time

[ 1.321054] md: linear personality registered for level -1
[ 1.326629] md: raid0 personality registered for level 0
[ 1.331964] md: raid1 personality registered for level 1
[ 1.342289] TCP cubic registered
[ 1.345936] NET: Registered protocol family 1
[ 1.350403] NET: Registered protocol family 17
[ 1.366034] drivers/rtc/hctosys.c: unable to open rtc device (rtc0)
[ 2.880506] IP-Config: Complete:
[ 2.883575] device=eth0, addr=192.168.1.201, mask=255.255.255.0, gw=255...
[ 2.892227] host=8349itx, domain=, nis-domain=(none),
[ 2.897798] bootserver=192.168.1.9, rootserver=192.168.1.9, rootpath=
initcall_debug

Great way to get a coarse view of system init timing

4 msecs: initcall c02c7da0 t linear_init
5 msecs: initcall c02c6bbc t init_sd
6 msecs: initcall c02c7d78 t mpc83xx_wdt_init
7 msecs: initcall c02cc450 t init_sunrpc
10 msecs: initcall c02c01a8 t slab_sysfs_init
15 msecs: initcall c02c8d50 t genl_init
24 msecs: initcall c02c55c4 t serial8250_init
30 msecs: initcall c02c6364 t gfar_init
34 msecs: initcall c02c743c t physmap_init
72 msecs: initcall c02c9c60 t inet_init
127 msecs: initcall c02c4e4c t pty_init
4597 msecs: initcall c02cabe0 t ip_auto_config
Example of boot time reduction

- i.MX31 Reference board

- Initial configuration:
  - 0:36 power on to command prompt
  - Redboot bootloader takes 10 seconds

- Final Configuration
  - Networking, static IP
  - Busybox userland
  - Many kernel optimizations
  - 2.7 Seconds - Kernel start to shell prompt
  - Minimal engineering investment
Applications/Userland

- **Startup scripts:**
  - Avoid Perl/Python dependencies (will reduce FS size as well)
  - One custom startup script instead of using SysV Init and /etc/init.d

- **Caches (big impact if you have a big userland):**
  - Library Cache setup
  - ICON
  - Font

- **Prelink libraries**

- **Use tools to profile execution**
  - strace and ltrace can be useful here
• Parallelize the init tasks
  • “boot” everything at once
  • drivers and tasks can do their several seconds of waiting at the same time

• Provide user feedback early
  • Splash screens, etc
  • give impression that unit is booted while initialization continues
  • Gets back to defining what “booted” means...
Save a canned memory and system state to non-volatile storage
- so that booting can occur as fast as the memory image can be read from disk (hibernation).
- Requires significant storage

Suspend/Resume – many consumer devices almost never cold boot.
- e.g. many “Smart Phones” only cold boot if you remove & replace battery
Oprofile – Systemwide Stat. Profiling for Linux
- On one project, this helped ID Flash as a bottleneck
- Focused on accelerating Flash I/O reduced boot time ~25%

http://elinux.org/Boot_Time

Tim Bird's OLS 2004 presentation
- Methods to Improve Bootup Time in Linux

Arjan van de Ven, Linux Plumbers Conference
Booting Linux in 5 Seconds (x86/Desktop focused)
(Sept 18, 2008)
  http://lwn.net/Articles/299483/

Meld (next slide)
A Meld discussion on Linux startup...
Thank you!

Chris Hallinan, Field Apps Engineer
MontaVista Software, Inc.
challinan@mvista.com