Reducing boot time in Linux devices

Chris Simmonds
2net Limited

Class WE-2.2
Embedded Live Conference 2010

EmbeddedLIVE
London • 19-21 October, 2010
Earls Court

Overview

- Users expect devices to be operational "immediately"
- But, complex operating systems such as Linux take time to boot
- In this presentation I will quantify the problem and look at methods to reduce boot time

Example system

Digi ConnectCore Wi-i.MX51

Hardware:
- Freescale i.MX515 @ 800 MHz
- 512 MiB SDRAM
- 512 MiB NAND flash
- Touch screen: 800x480x16

Software:
- U-Boot
- Linux 2.6.31
- Ångström root file system
- jffs2 flash file system
- Simple camera app in Qte4

Understanding the problem

- How to measure boot time?
  - stop watch?
  - monitor console output?
  - instrument the code?

grabserial

- http://elinux.org/Grabserial
- Python script that adds a time stamp to serial
  - Written by Tim Bird
- Captures whole boot sequence from power-on to running application
- We have a serial console, so let's try it out!

Using grabserial

Usage:

    -d <serial device>
    -b <baudrate>
    -m <match pattern that will reset time stamps>
    -t

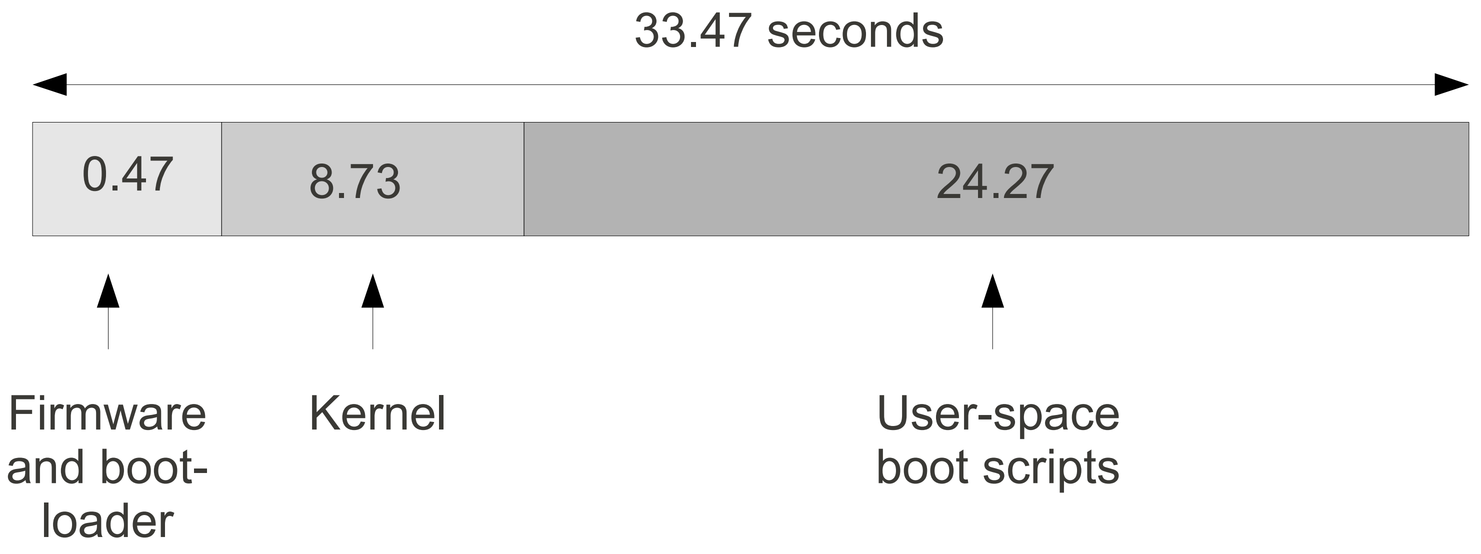
For example:

```
grabserial -t -d /dev/ttyUSB0 -b 38400 -m "Starting*"

[    7.463323] Starting kernel ...
[    0.009875]
[    0.010001] Uncompressing Linux...............................................
................................................................................
....... done, booting the kernel.
[    1.099339] Linux version 2.6.31 (chris@chris-laptop) (gcc version 4.3.3 (GCC)
 ) #1 PREEMPT Wed Sep 29 17:03:06 BST 2010
[    1.126183] CPU: ARMv7 Processor [412fc085] revision 5 (ARMv7), cr=10c53c7f
[    1.142990] CPU: VIPT nonaliasing data cache, VIPT nonaliasing instruction cache
[    1.159425] Machine: Digi ConnectCore Wi-MX51 on a JSK Board
```

First pass

- Boot-up time from power-on to usable device is composed of



Analysis of user space boot

- Here is a list of tasks taking more than 220 milliseconds:

Task	Time
Starting Avahi mDNS/DNS-SD Daemon	20.2675
Starting Bluetooth subsystem	1.1901
Starting udevd	0.5276
Remounting root file system	0.2200

Removing services

- Network services are often quite slow to set-up
  - in this case Avahi mDNS: we don't need it at all
  - if you really need the service, maybe you can start it later after the device is operational
- Other services we can get rid of
  - Bluetooth
  - remounting root file system: just not necessary with jffs2

udev

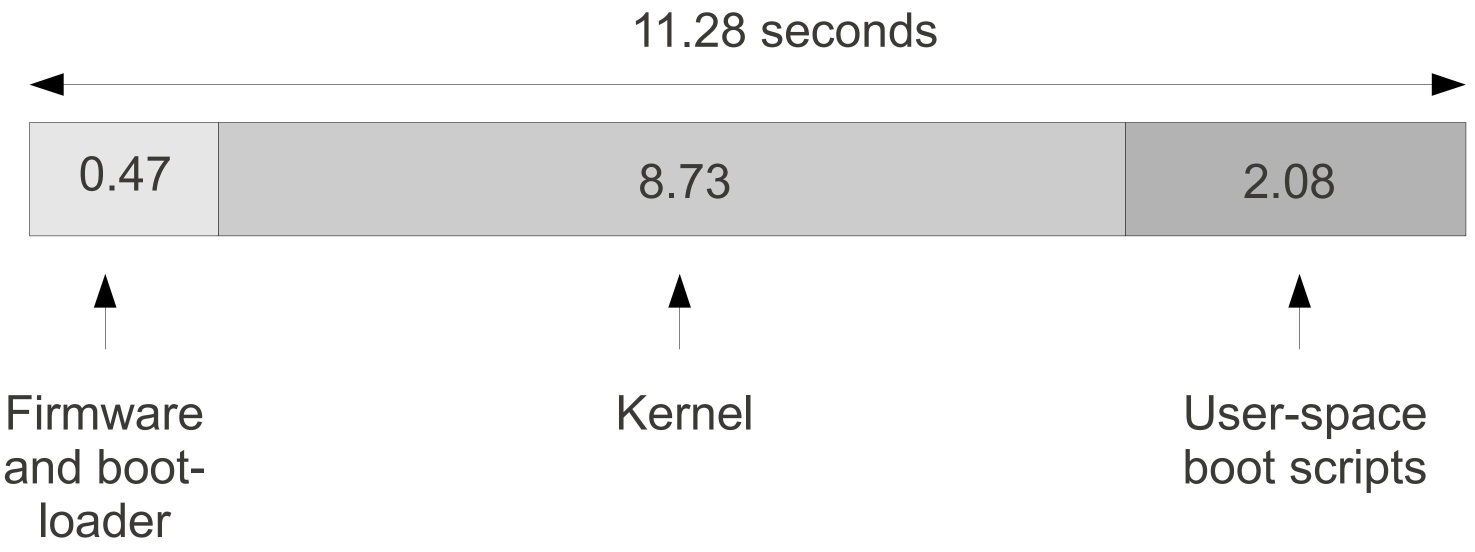
- Populates dev with device nodes
- Responds to run-time events
  - adding/removing hardware
  - loading/unloading modules
  - creating/deleting device nodes
- Maybe we don't need it at all?
  - many devices have a static or well-known set of devices
- Or, maybe we can achieve the same thing another way?

No udev

- Create device nodes my hand (quite tedious):
    - mknod -m 666 /dev/null c 1 3
    - etc, etc, etc...
- Or,
    - boot with udev
    - create a tar archive of /dev
    - extract over /dev on the master copy
    - disable udev
    - re-flash the root file system

After optimised user startup:

- Saving: 22.19 seconds



Measuring kernel boot: PRINTK_TIME

- Enable in Kernel Hacking->Show timing information on printks
- Adds a time stamp to kernel printk
- Similar to grabserial: useful if you don't have a serial console

```
[    0.000000] Linux version 2.6.31 (chris@chris-laptop) (gcc version 4.3.3 (GCC
) ) #2 PREEMPT Wed Sep 29 17:59:01 BST 2010
[    0.000000] CPU: ARMv7 Processor [412fc085] revision 5 (ARMv7), cr=10c53c7f
[    0.000000] CPU: VIPT nonaliasing data cache, VIPT nonaliasing instruction ca
che
[    0.000000] Machine: Digi ConnectCore Wi-MX51 on a JSK Board

...
```

Mounting jffs2

- One section stands out in the kernel log:

```
[   10.340000] JFFS2 doesn't use OOB.
[   12.960000] VFS: Mounted root (jffs2 filesystem) on device 31:3.
```

Mount time 2.62 s for a 470 MiB jffs2 file system which is 8% full

Options to speed up root mount

- Make the root partition smaller
  - Only using 8%: smaller is faster
- Use UBIFS
  - a more modern (& faster) flash file system
- Use a read-only file system, e.g. squashfs
  - need a separate read/write partition to store data
  - squashfs requires a UBI volume to cope with NAND flash

Using ubifs

UBI attach takes 1.39s

```
[    2.240000] UBI: attaching mtd3 to ubi0
...
[    3.620000] UBI: background thread "ubi_bgt0d" started, PID 440
[    3.630000]
...
[    5.670000] UBIFS: mounted UBI device 0, volume 0, name "ubi_rfs"
[    5.780000] VFS: Mounted root (ubifs filesystem) on device 0:12.
```

UBIFS mount takes 0.11s

Total: 1.50s: **a saving of 1 second** over JFFS2
Note: there would be a larger saving with more files in the root file system

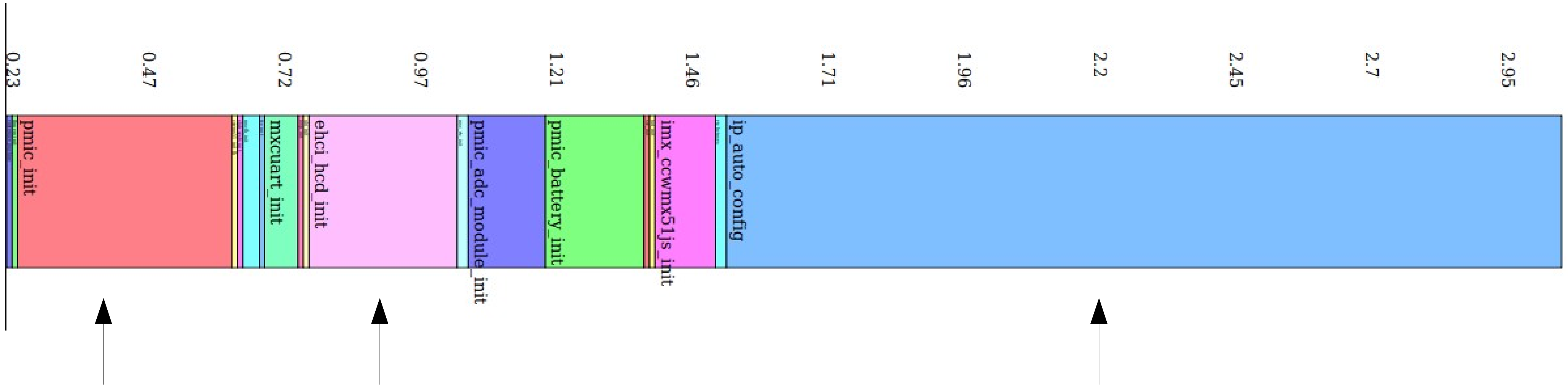
Quiet boot

- All those strings printed over the serial port take time
- Add “quiet” to bootargs
    - before 5.80s
    - after 2.73s
    - **saving: 3.07 seconds**

Kernel boot-tracer

- A more sophisticated boot time analyser
  - Enable in Kernel Hacking-> Tracers-> Trace boot initcalls
  - Requires 2.6.28 kernel
  - Increase kernel log buffer size to 16 (64KB)
- Boot with "initcall_debug printk.time=1"
- Then,
  - dmesg -s 65536 > /boot.log
- Copy boot.log to your PC, and
  - cat boot.log | perl linux-2.6.31/scripts/bootgraph.pl > bootgraph.svg

Boot trace output



pmic_init
0.38 s

ehci_hcd_init
0.26 s

ip_auto_config
1.48 s

$\underbrace{\hspace{10em}}_{\text{un-avoidable}}$

ip_auto_config

- Allows setting IP address on kernel command line
- Not needed in production systems
  - **Saving: 1.48 seconds**

Loops-per-jiffy

```
[    0.000000] Calibrating delay loop... 799.53 BogoMIPS (lpj=3997696)
[    0.230000] Mount-cache hash table entries: 512
```

230ms taken to calculate the lpj figure of 3997696, which will always be the same!

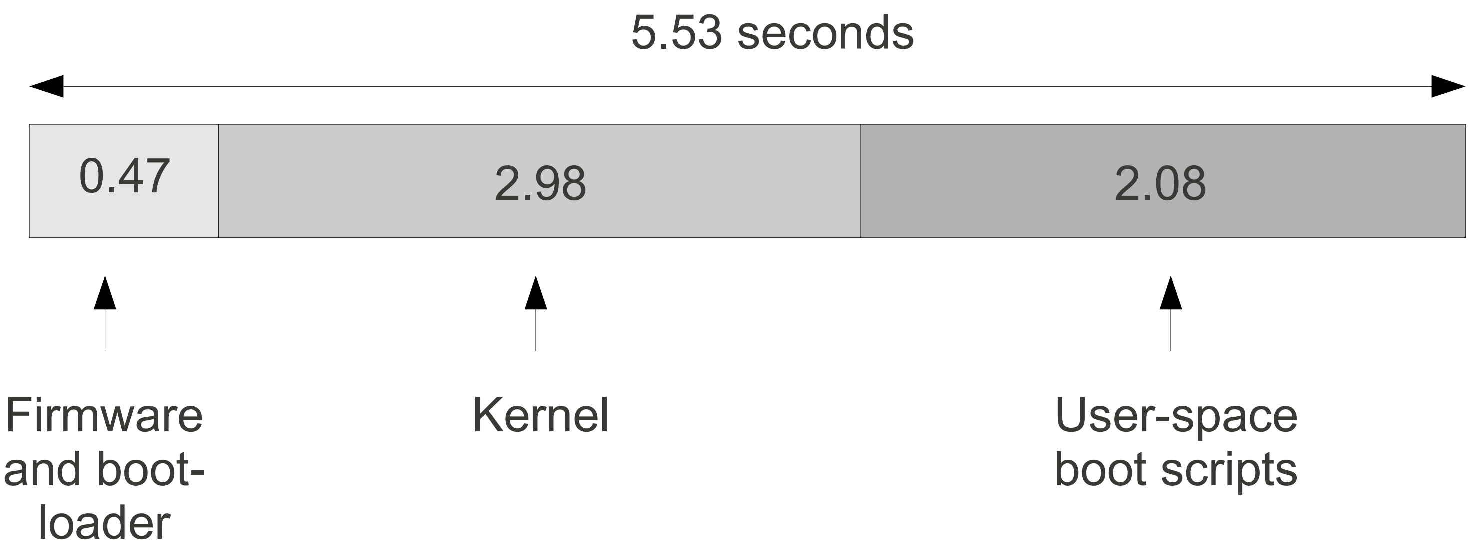
Set in the kernel command line:

```
    console=ttymxc1,38400 quiet lpj=3997696
```

Saving: 0.23 seconds

After kernel optimisation

- Boot time just over 5 seconds
  - which is acceptable!



Other strategies

- A boot time of 5 seconds is acceptable in this case
- I could have continued the process
    - but it gets harder...
- The next few slides present some ideas

Reduce kernel size

- Less code to load and decompress
- Remove unnecessary drivers
  - and unnecessary driver initialisation code
- Configure drivers not essential to boot as modules & load them later

Optimise boot loader

- In some systems the boot loader may be a significant delay
- Typical areas to consider
  - Instruction and data caches turned on?
  - Lengthy or unnecessary probing of devices?
  - Verbose message to serial console?

Kernel loading time

- On a slow CPU the time to decompress the kernel image can be significant
  - Store the kernel uncompressed
- Use a DMA channel to copy to memory

Summary

- Device boot times need not be multiple 10's of seconds
- In user space:
  - optimise the boot scripts
- In the kernel:
  - Reduce kernel verbosity with “quiet”
  - Choose the right file systems

Links

- Inner Penguin blog at
  - http://www.embedded-linux.co.uk
- 2net web site
  - http://www.2net.co.uk
- Embedded Linux Wiki
  - http://elinux.org/Boot_Time