Techniques for Improving Embedded Linux Boot Times

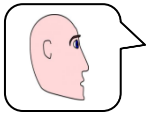
This slide is replaced by Open Systems Media

Agenda

- **Why Fastboot?**
- **The Boot Sequence**
- **System Instrumentation**
- **Optimizations**
    - Bootloader
    - Kernel
    - Middleware & Applications
- **Customer Example – One Second Boot**

Why Optimize the Bootup?

- **First , faster is always better if you ask the customer:**



”How fast do you need it to be?”

”As fast as possible!”

  - Optimization is not necessarily a huge & time consuming task

- **Especially important for products meant for the consumer market.**
  - User impatience = competitive edge
  - Cell phones, PDAs, MP3 players, generally systems with a (G)UI

- **Critical applications**
  - Reboot in Carrier Grade Systems – High Availability
  - System watchdog fires in a critical system (medical, security, industrial control) – must be back online ”instantly”!

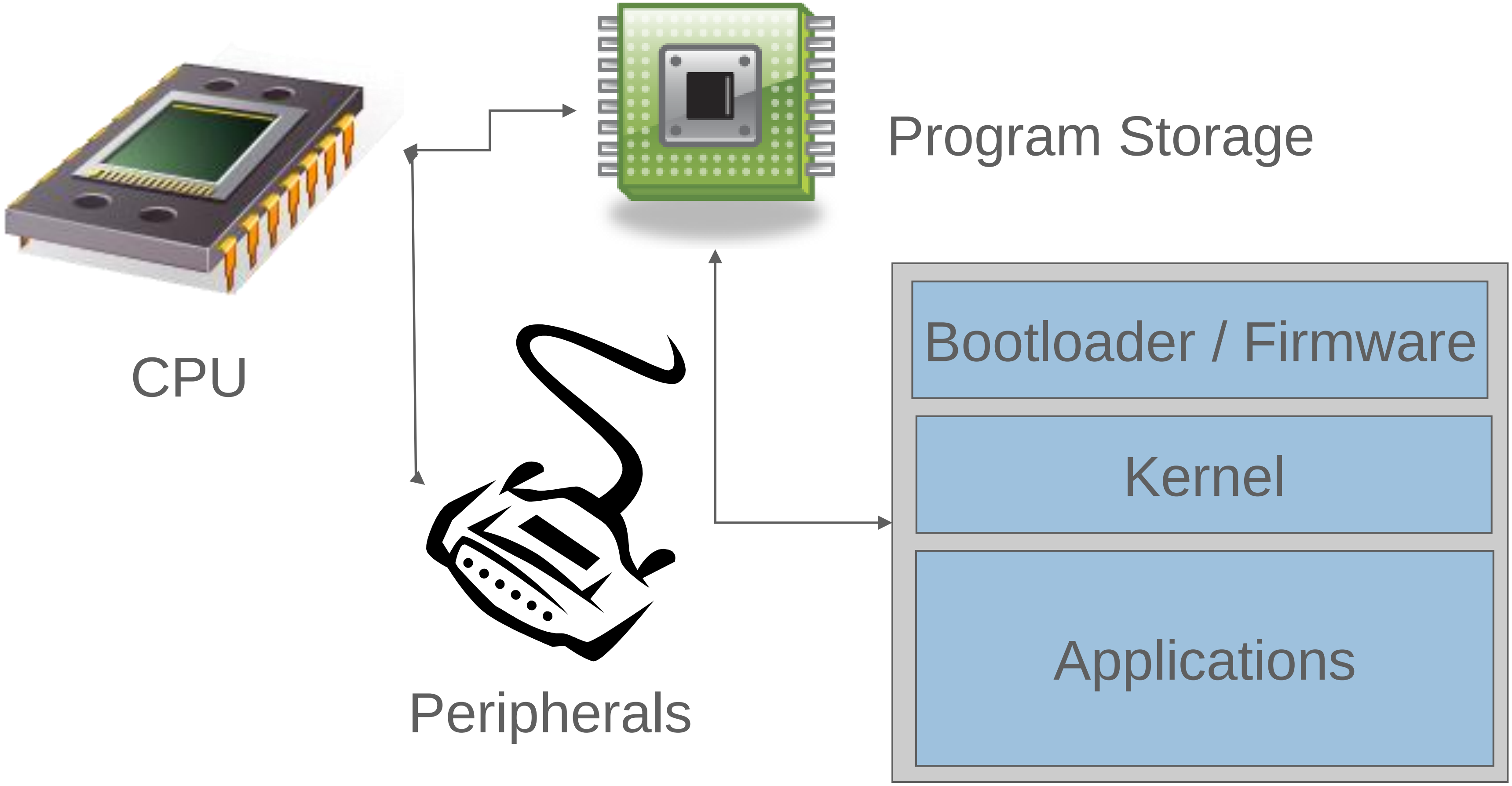
Agenda

- Why Fastboot?
- **The Boot Sequence**
- System Instrumentation
- Optimizations
  - Bootloader
  - Kernel
  - Middleware & Applications
- Customer Example – One Second Boot

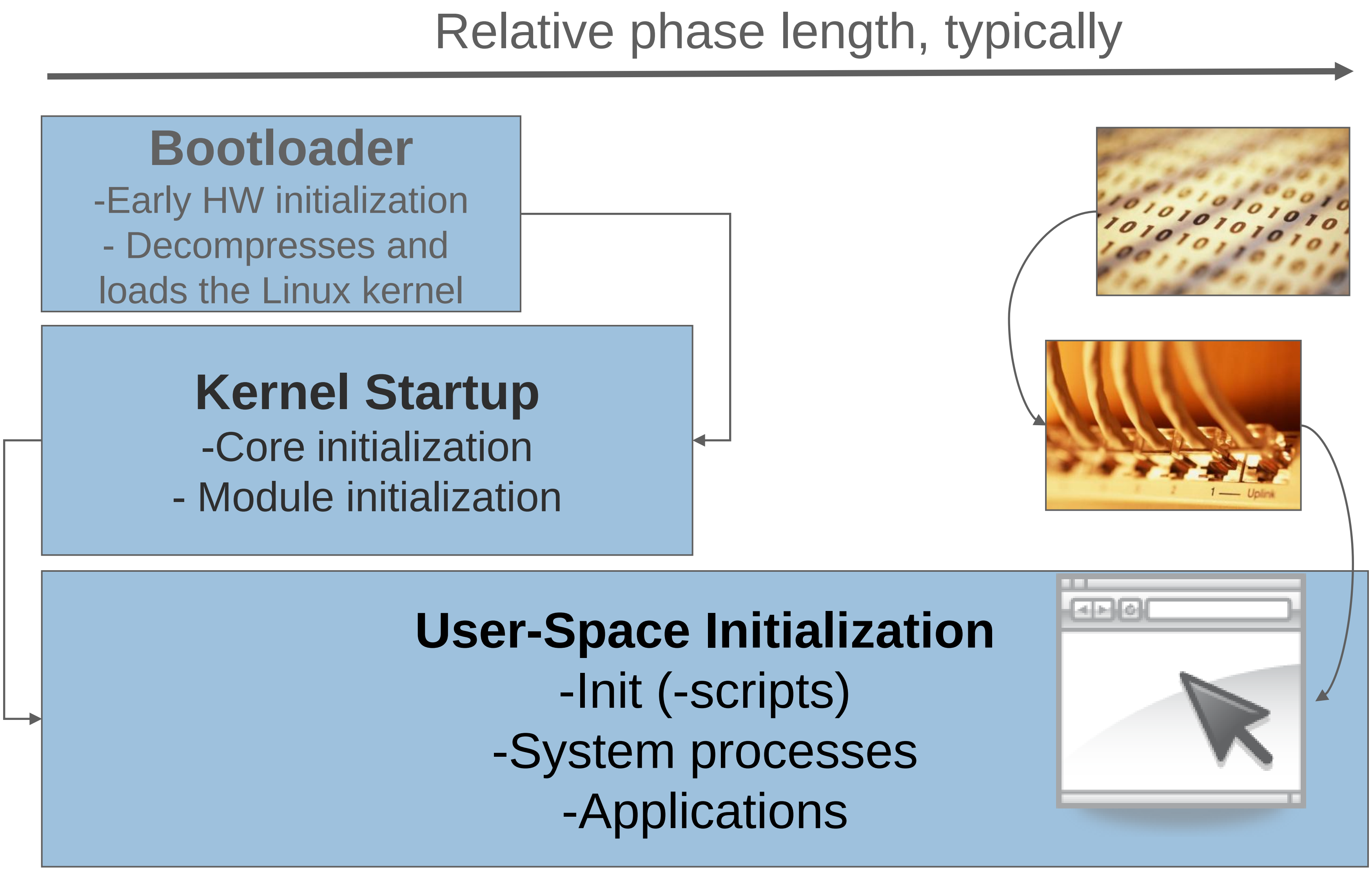
© 2010 MontaVista Software

Bootup Time Components – Holistic View

Getting the product to boot quickly is dependent on the total system design!



The Bootup Phases



Relative phase length, typically

Bootloader

-Early HW initialization

- Decompresses and loads the Linux kernel

Kernel Startup

-Core initialization

- Module initialization

User-Space Initialization

-Init (-scripts)

-System processes

-Applications

Agenda

- Why Fastboot?
- The Boot Sequence
- System Instrumentation
- Optimizations
  - Bootloader
  - Kernel
  - Middleware & Applications
- Customer Example – One Second Boot

Uptime

- A very simple utility to do timekeeping
- Outputs the time since clock initialized and the time spent running the idle process

```
Uptime 66.54 45.23
```

- Put everywhere you want to timestamp!

Timing Printk

- **A simple method to put a timestamp on every printk**
- **Activation (use one of the following):**
  - Compile kernel with: CONFIG_PRINTK_TIMES=y (in Kernel Hacking)
  - Use “time” on kernel command line (or for later kernels printk.time = 1/Y/y)
  - Or, dynamically in a run-time system (as root):
    - “echo 1 >/sys/module/printk/parameters/printk_time”
    - “echo 1/Y/y >/sys/module/printk/parameters/time”

initcall_debug

- **The modules initialization calls, “initcalls” spend a considerable time on kernel bootup**

- **There’s a flag already built into the kernel to show initcall information during startup**

- **Activating: On the command line, add “initcall_debug=1”**

- **NOTE:**
  - increase the printk log buffer size in kernel config:
    - LOG_BUF_SHIFT=18 (256KB)
    - Remember to enable printk-times to get timing info!

- **After booting, do:**
  dmesg -s 256000 | grep "initcall" | sed "s/\(.*\)after\(.*\)/\2 \1/g" | sort -n

- **More info: http://elinux.org/Initcall_Debug**

Initcall_debug Example Output

```
24 msecs [    2.237177] initcall acpi_button_init+0x0/0x51 returned 0
28 msecs [    0.763503] initcall init_acpi_pm_clocksource+0x0/0x16c returned 0
32 msecs [    0.348241] initcall acpi_pci_link_init+0x0/0x43 returned 0
33 msecs [    0.919004] initcall inet_init+0x0/0x1c7 returned 0
33 msecs [    5.282722] initcall psmouse_init+0x0/0x5e returned 0
54 msecs [    2.979825] initcall e100_init_module+0x0/0x4d returned 0
71 msecs [    0.650325] initcall pnp_system_init+0x0/0xf returned 0
91 msecs [    0.872402] initcall pcibios_assign_resources+0x0/0x85 returned 0
187 msecs [    4.369187] initcall ehci_hcd_init+0x0/0x70 returned 0
245 msecs [    2.777161] initcall serial8250_init+0x0/0x100 returned 0
673 msecs [    5.098052] initcall uhci_hcd_init+0x0/0xc1 returned 0
830 msecs [    4.067279] initcall piix_init+0x0/0x27 returned 0
1490 msecs [    8.290606] initcall ip_auto_config+0x0/0xd70 returned 0
```

- **Problem routines:**
  - psmouse_init - unused driver!!
  - pnp_system_init, pcibios_assign_resources- ??
  - ehci_hcd_init, uhci_hcd_init - part of USB initialization
  - serial8250_init - serial driver initialization
  - piix_init – IDE disk driver init
  - ip_auto_config - dhcp process

grabserial

- **A python utility for watching serial console output**
  - Requires the python serial module (non-default)
- **The tool is run on the host**
  - Basically reads the serial input and pushes to stdout
  - Doesn’t slow down the target
- **But it can add timing information on the output!**
  - Allows bootup timing
- **See also ”show_delta” in linux_src/scripts/show_delta**
  - Adds timing delta info
- **Easy to use**
  - Ex: grabserial –t –d /dev/ttyUSB0 –m “Starting kernel”

grabserial - example

```
lappis:/home/iiskol # grabserial -t -e 60
[    0.000001] OK
[    1.171282] OK
[    1.175303]
[    1.175364] Starting kernel ...
[    1.175946]
[    1.184604] Uncompressing
Linux.........................                    Ft Linux version
2.6.24_mvl5024-omap3530_evm-iisko (iiskol@lappis) (gcc version 4.2.0
(MontaVista 4.2.0-16.0.25.0801369 2008-06-27)) #7 PREEMPT Wed Nov 19
14:50:33 EET 2008
[   24.605721] CPU: ARMv7 Processor [411fc082] revision 2 (ARMv7),
cr=00c5387f
[   24.612978] Machine: OMAP3EVM Board
[   24.613689] Memory policy: ECC disabled, Data cache writeback
[   24.618029] OMAP3430 ES2.2
[   24.621541] SRAM: Mapped pa 0x40200000 to va 0xd7000000 size:
0x100000
[   24.625632] CPU0: D VIPT write-through cache
[   24.629457] CPU0: cache: 768 bytes, associativity 1, 8 byte lines, 64
sets
[   24.639515] Built 1 zonelists in Zone order, mobility grouping on.
Total pages: 32512
[   24.641956] Kernel command line: root=/dev/nfs rw
nfsroot=192.168.2.2:/media/linux_backup/Dev-Area/armv7_le_root
ip=192.168.2.1 console=ttyS0,115200n8
[   24.655200] GPMC revision 5.0
[   24.655711] IRQ: Found an INTC at 0xd8200000 (revision 4.0) with 96
interrupts
[   24.663604] Total of 96 interrupts on 1 active controller
[   24.665501] OMAP34xx GPIO hardware version 2.5
[   24.669136] PID hash table entries: 512 (order: 9, 2048 bytes)
[   24.673234] Console: colour dummy device 80x30
[   24.678639] Dentry cache hash table entries: 16384 (order: 4, 65536
bytes)
[   24.681688] Inode-cache hash table entries: 8192 (order: 3, 32768
bytes)
[   24.690468] Memory: 128MB 0MB = 128MB total
[   24.691493] Memory: 122624KB available (6276K code, 743K data, 172K
init)
[   24.697067] Security Framework initialized
[   24.698034] Capability LSM initialized
[   24.701668] Mount-cache hash table entries: 512
[   24.705450] CPU: Testing write buffer coherency: ok
[   24.709358] net_namespace: 76 bytes
[   24.710189] NET: Registered protocol family 16
[   24.714027] I2C Client[3] is not initialized[511]
[   24.717378] I2C Client[3] is not initialized[460]
[   24.721131] SmartReflex driver initialized
[   24.726020] OMAP DMA hardware revision 4.0
[   24.726523] Initializing OMAP McBSP system
[   24.729423] USB: No board-specific platform config found
[   24.734463] OMAP Display hardware version 2.0
[   24.737487] i2c_omap i2c_omap.1: bus 1 rev3.12 at 2600 kHz
[   24.741338] i2c_omap i2c_omap.2: bus 2 rev3.12 at 100 kHz
[   24.745367] i2c_omap i2c_omap.3: bus 3 rev3.12 at 400 kHz
[   24.749595] TWL4030: TRY attach Slave TWL4030-ID0 on Adapter OMAP I2C
adapter [1]
[   24.757229] TWL4030: TRY attach Slave TWL4030-ID1 on Adapter OMAP I2C
adapter [1]
[   24.763684] TWL4030: TRY attach Slave TWL4030-ID2 on Adapter OMAP I2C
adapter [1]
[   24.769225] TWL4030: TRY attach Slave TWL4030-ID3 on Adapter OMAP I2C
adapter [1]
[   24.773667] TWL4030 Power Companion Active
[   24.777413] <6>TWL4030: Driver registration complete.
[   24.781392] TWL4030 GPIO Demux: IRQ Range 384 to 402, Initialization
Success
[   24.788900] Initialized TWL4030 USB module
[   24.789546] usbcore: registered new interface driver usbfs
[   24.793560] usbcore: registered new interface driver hub
[   24.797597] usbcore: registered new device driver usb
[   24.802446] Time: 32k_counter clocksource has been installed.
```

strace

- **Strace can be used to collect timing information for a process**
  - strace –tt 2>/tmp/strace.log thttpd …
- **Can use to see where time is being spent in application startup**
- **Can also collect system call counts (-c)**
- **Can see time spent in each system call (-T)**
- **Great for finding extraneous operations**
  - scanning invalid paths for files,
  - opening a file multiple times, etc.
- **Strace can follow children**
- **Strace adds of overhead to the execution of the program**
  - Good for relative timings, not absolute
- **Can’t get counts for a program that doesn’t end**

strace Example Output

```
00:00:07.186340 mprotect(0x4001f000, 20480, PROT_READ|PROT_WRITE) = 0
00:00:07.200866 mprotect(0x4001f000, 20480, PROT_READ|PROT_EXEC) = 0
00:00:07.221679 socketcall(0x1, 0xbe842c70) = 3
00:00:07.235626 fcntl64(3, F_SETFD, FD_CLOEXEC) = 0
00:00:07.248718 socketcall(0x3, 0xbe842c70) = -1 EPROTOTYPE (Protocol wrong type
 for socket)
00:00:07.264434 close(3)                = 0
00:00:07.286956 socketcall(0x1, 0xbe842c70) = 3
00:00:07.292816 fcntl64(3, F_SETFD, FD_CLOEXEC) = 0
00:00:07.305603 socketcall(0x3, 0xbe842c70) = 0
00:00:07.327575 brk(0)                  = 0x24000
00:00:07.345397 brk(0x25000)            = 0x25000
00:00:07.360290 brk(0)                  = 0x25000
00:00:07.422485 open("/etc/thttpd/thttpd.conf", O_RDONLY) = 4
00:00:07.438049 fstat64(4, {st_mode=S_IFREG|0644, st_size=17592186044416, ...})
= 0
00:00:07.474121 old_mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONY
MOUS, -1, 0) = 0x40017000
00:00:07.490203 read(4, "#-------------------------------"..., 4096) = 1457
00:00:07.508544 read(4, "", 4096)       = 0
00:00:07.530151 close(4)                = 0
00:00:07.548675 munmap(0x40017000, 4096) = 0
00:00:07.561645 open("/etc/localtime", O_RDONLY) = -1 ENOENT (No such file or di
rectory)
00:00:07.585235 open("/etc/thttpd/throttle.conf", O_RDONLY) = 4
00:00:07.599182 gettimeofday({7, 603149}, NULL) = 0
00:00:07.613983 fstat64(4, {st_mode=S_IFREG|0644, st_size=17592186044416, ...})
= 0
00:00:07.637084 old_mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONY
MOUS, -1, 0) = 0x40017000
00:00:07.650604 read(4, "# thttpd 2.21b\n# Main throttle c"..., 4096) = 453
00:00:07.669586 read(4, "", 4096)       = 0
00:00:07.691589 close(4)                = 0
00:00:07.708099 munmap(0x40017000, 4096) = 0
```

ftrace

- **Mainlined in 2.6.27**
- **Derived from RT-preempt latency tracer**
- **Instrumentation**
  - Explicit
    - Tracepoints defined by declaration
    - Calls to trace handler written in source code
  - Implicit
    - Automatically inserted by compiler
      - Uses gcc ‘-pg’ option
      - Only traces function entry

Using ftrace early in boot sequence

- **Use “ftrace=function_duration” on kernel command line**
- **Tracer is initialized after kernel core (timers, memory, interrupts), but before all initcalls**
- **Need to stop trace after point of interest**
  - Use “trace_stop_fn=<func_name>” on kernel command line
  - Trace stops on ENTRY to named function
  - An initcall works very well
    - Use one that is called immediately after your area of interest
- **Overhead of ftrace can be big**
  - Average function duration = 3.22 µs
  - Overhead = 11.4 microseconds per function

fdd tool output

Total time may be wrong if process is scheduled out or if a filter was active

```
$ fdd /tmp/trace.txt -n 15
Function                            Count Time       Average  Local
----------------------------------- ----- ---------- -------- ----------
schedule                               59 1497735270 25385343 1476642939
sys_write                              56 1373722663 24530761    2892665
vfs_write                              56 1367969833 24428032    3473173
tty_write                              54 1342476332 24860672 1212301170
do_path_lookup                         95 1076524931 11331841   34682198
__link_path_walk                       99 1051351737 10619714    6702507
rpc_call_sync                          87 1033211085 11875989    1700178
path_walk                              94 1019263902 10843233    3425163
rpc_run_task                           87  960080412 11035407    2292360
rpc_execute                            87  936049887 10759194    2316635
__rpc_execute                          87  932779083 10721598   11383353
do_lookup                             191  875826405  4585478    9510659
call_transmit                         100  785408085  7854080    5871339
__nfs_revalidate_inode                 38  696216223 18321479    1652173
nfs_proc_getattr                       38  690552053 18172422    1234634
```

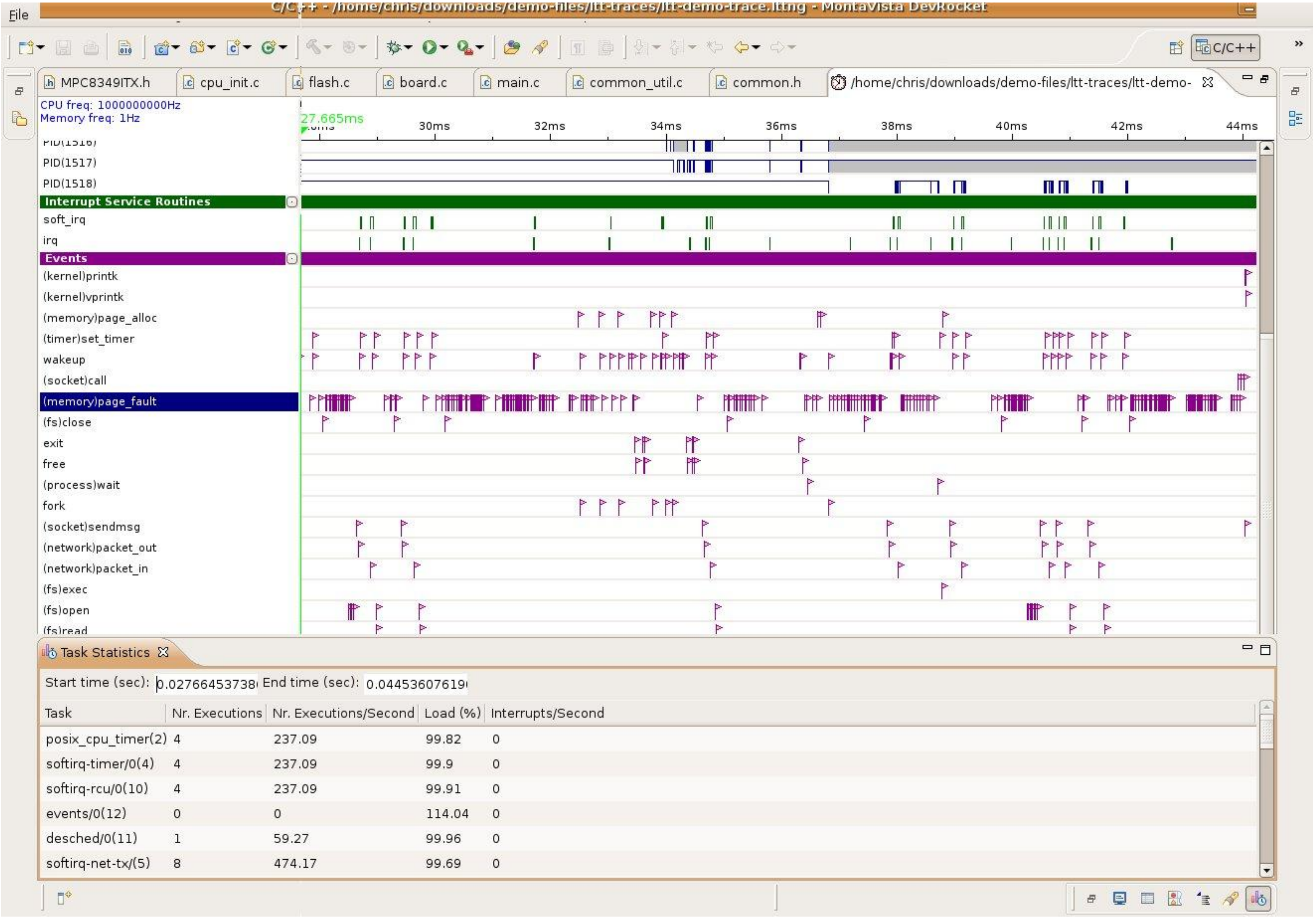
Want more?

- **Tim Bird's presentation on ftrace**
  - *www.elinux.org/images/e/e8/Bird-Ftrace.ppt*
- **Ftrace tutorial at OLS 2008**
  - http://people.redhat.com/srostedt/ftrace-tutorial.odp
- **"The world of Ftrace" at Spring 2009 LF Collaboration Summit**
  - http://people.redhat.com/srostedt/ftrace-world.odp
- **Patches and tools for this talk**
  - http://elinux.org/Ftrace_Function_Graph_ARM

Linux Trace Toolkit

- **Traces almost everything going on in the system**
- **Especially good for debugging process interaction, startup times and race-conditions**
- **Configurable**
- **Supported by MontaVista**
- **Integrated into DevRocket**
- **Not supported in Main Line**
  - See http://lttng.org/

Linux Trace Toolkit



Start time (sec): 0.02766453738 End time (sec): 0.04453607619

Task	Nr. Executions	Nr. Executions/Second	Load (%)	Interrupts/Second
posix_cpu_timer(2)	4	237.09	99.82	0
softirq-timer/0(4)	4	237.09	99.9	0
softirq-rcu/0(10)	4	237.09	99.91	0
events/0(12)	0	0	114.04	0
desched/0(11)	1	59.27	99.96	0
softirq-net-tx/(5)	8	474.17	99.69	0

Boot_tracer

- **To Enable (on 2.6.28 or later kernels)**
  - CONFIG_BOOT_TRACER
  - depends on: CONFIG_DEBUG_KERNEL
- **Records the timings of the initcalls and traces key events and the identity of oiffending tasks**
- **Targeted to be parsed by the /scripts/bootgraph.pl tool to produce pretty graphics about boot inefficiencies**
- **Raw /debug/tracing/trace text output is readable too.**

- **And…SystemTap**
  - Requires kernel loadable modules
  - Requires module insertion (user space must be up)

Agenda

- Why Fastboot?
- The Boot Sequence
- System Instrumentation
- **Optimizations**
  - Bootloader
  - Kernel
  - Middleware & Applications
- Customer Example – One Second Boot

Typical Bootloader

- **Lots of useful development functionality**
  - tftp, pci scan, mem utils
  - device initialization, etc
- **In a production system, many of these features are unnecessary**
- **Disabling these features can have a significant impact on boot time**
- **You want the bootloader to do it's work and get out of the way as fast as possible**

”Splash Screens”

- **Indicates system is active, but still booting**

- **A splash screen can take place**
  - In the bootloader
    - http://www.denx.de/wiki/DULG/UBootSplashScreen
    - Can also pre-initialize HW, removing the need in Linux
    - Needs kernel customization

  - In the kernel
    - After initialization of the framebuffer driver

  - Early user-space init
    - Custom, but before system apps initialized

Bootloader Optimization

- **We did this example using U-Boot, but the same techniques are applicable to other bootloaders**

- **Major Time Consumers**

Relocation from Flash to RAM	1.3s
PCI Initialization	1.0s
IDE Initialization	1.2s
Board specific devices	0.4s

- **After optimization boot time dropped from 4.25s to 1.1s**
- **A 75% reduction in time!**

Details of U-Boot Optimizations

- **Disabled “compare” operation on U-Boot copy**
- **Removed support for PCI and IDE**
  - Could avoid lengthy bus scan by “hard-coding” fixed configuration
  - Could also enable a no-probe mode
- **Used some config options for faster boot**
  - **CFG_CONSOLE_INFO_QUIET**
    - Suppress display of console information at boot
  - **#undef CONFIG_PCI_SCAN_SHOW**
    - Suppress display of PCI devices
  - **#undef CONFIG_SPD_EEPROM**
    - Fixed RAM configuration instead of SPD
  - **CONFIG_PCI**
    - Speeds up boot if not using PCI
    - Could also optimize to eliminate scan
  - **CONFIG_IDE**
    - Eliminate if not using IDE

Linux Kernel Configuration

- **Eliminate Unnecessary Kernel Options**
  - Reduces kernel size
  - Speeds up kernel loading

- **Typical default kernel config contains lots of 'stuff' you may not need**
  - MD/Raid support, IPv6, Numerous File Systems, Extended Partition support, etc.
  - Debug features such as kernel symbols, kconfig, etc.
  - Many are compiled in features and increase kernel size

Kernel Config Options

- **CONFIG_IKCONFIG**
  - Removes support for config info, makes kernel smaller (~ 250 ms improvement)
- **CONFIG_MD**
  - RAID/LVM support
- **CONFIG_IDE**
  - Saves init time if not used
  - Can also use hdx=noprobe
- **CONFIG_DEBUG_KERNEL**
  - Reduces kernel size substantially
- **CONFIG_KALLSYMS**
  - Different than gcc –g
- **CONFIG_PCCARD**
  - Disable PCMCIA if not required
- **Check Networking config options**
  - Lots of functionality there, do you need it all?
  - SCTP, TIPC, etc.

More Kernel Config Options

- **CONFIG_HOTPLUG**
  - Remove support for hotplug if not required
- **CONFIG_BUG**
  - Used for debug – can be removed if desired
- **Check Device Driver config options**
  - Lots of default functionality that you may not need
- **Anything compiled as a module, if unused, is irrelevant**
  - Won't affect startup time
- **Remove support for unnecessary File System features**
  - Default configs often have much of this enabled (=y)
    - CONFIG_DNOTIFY
    - CONFIG_INOTIFY
    - CONFIG_XFS
    - CONFIG_AUTOFS4_FS (Automounter)
  - Won't make a large performance difference, but a smaller kernel will definitely load faster
  - 18.5% smaller after removing unused FS features!

XIP – Execute in Place

- **Processor does not copy Kernel image to DRAM**
  - Executes directly from (NOR) Flash
- **Advantages**
  - Reduces amount of DRAM required
  - 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
- **To reduce userspace application load times:**
  - New flash filesystem: AXFS

Calibration Routines

- **Many hardware platforms spend considerable time in calibration routines**
  - Allows precise delay() routines
  - Can take significant time
- **Use kernel command line loops-per-jiffy**
  - lpj=xxxx
  - Easy to use, most platforms will display the correct value in the boot log on startup

Driver Configuration

- **Consider your system requirements**
  - What functionality must be available immediately?
  - What functionality can be deferred?
- **Pass device parameters in firmware**
- **Drivers can be pre-compiled into the kernel or made as modules for loading later**
  - Use pre-compiled drivers for those functions that must be immediately available
  - Use loadable modules for deferred functions

Patch to shorten Network Init

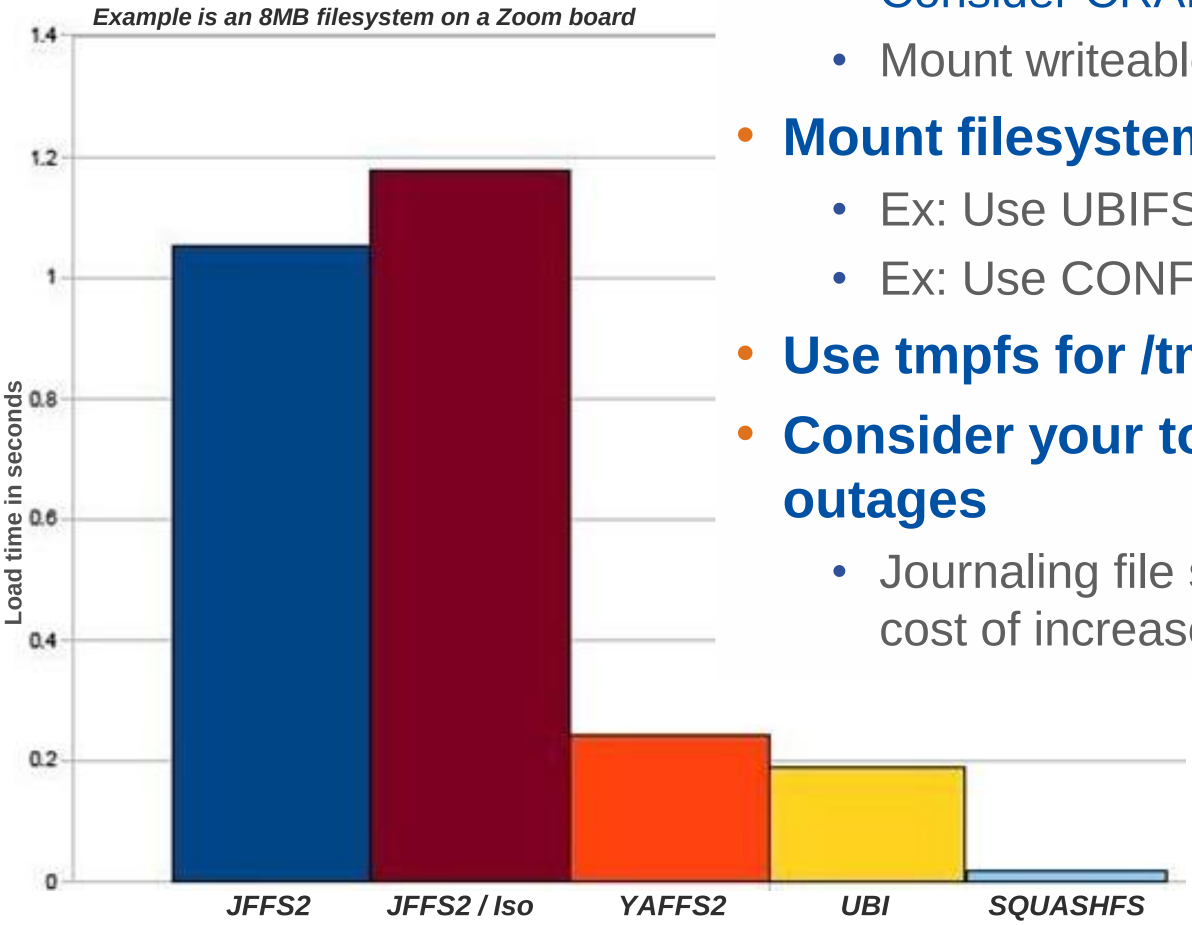
- **Generic mainline code has to work with every conceivable piece of hardware**
- **Delays are often too long for specific hardware**

```
diff --git a/net/ipv4/ipconfig.c b/net/ipv4/ipconfig.c
index 42065ff..e42d83f 100644
--- a/net/ipv4/ipconfig.c
+++ b/net/ipv4/ipconfig.c
@@ -86,8 +86,10 @@
 #endif

 /* Define the friendly delay before and after opening net devices */
-#define CONF_PRE_OPEN          500     /* Before opening: 1/2 second */
-#define CONF_POST_OPEN         1       /* After opening: 1 second */
+/*#define CONF_PRE_OPEN                500     /* Before opening: 1/2 second */
+/*#define CONF_POST_OPEN               1       /* After opening: 1 second */
+#define CONF_PRE_OPEN          5       /* Before opening: 5 milli seconds */
+#define CONF_POST_OPEN         10      /* After opening: 10 milli seconds */

 /* Define the timeout for waiting for a DHCP/BOOTP/RARP reply */
 #define CONF_OPEN_RETRIES      2       /* (Re)open devices twice */
@@ -1292,7 +1294,7 @@ static int __init ip_auto_config(void)
                return -1;

        /* Give drivers a chance to settle */
-       ssleep(CONF_POST_OPEN);
+       msleep(CONF_POST_OPEN);

        /*
         * If the config information is insufficient (e.g., our IP address or
```

- **Patch shows reduction in delay for IP autoconfig**
- **X86 savings: 1.4 seconds**

File System Selection

- **Partition filesystem into read-only portion and read/write portion**
  - Read-only file systems mount faster
  - Consider CRAMFS for initial read-only FS
  - Mount writeable FS later, such as JFFS2
- **Mount filesystem faster:**
  - Ex: Use UBIFS/Squashfs
  - Ex: Use CONFIG_JFFS2_SUMMARY
- **Use tmpfs for /tmp, possibly /var, others**
- **Consider your tolerance to sudden power outages**
  - Journaling file systems can protect but at a cost of increased startup times



Courtesy of Free Electrons under Creative Commons BY-SA license
http://free-electrons.com/pub/conferences/2008/elce/flash-filesystems.pdf

Remove printk() Support

- **The “Brute Force” approach - CONFIG_PRINTK**
  - Completely eliminates calls to printk()
- **Advantages**
  - Saves significant kernel size, and therefore load time
  - Eliminates many boot message, therefore decreasing boot time
- **Disadvantage**
  - No kernel status message are available!
- **There is a “middle ground” for kernel development & debugging**
- **A well tested kernel should work in this mode**

Userspace - Optimize Init

- **A large issue in embedded systems: use BusyBox**
- **It is also possible to create a custom init program to run instead of normal init**
  - Linux looks for /sbin/init, /etc/init, /bin/init, /bin/sh. In that order.
  - Can configure: init = xxx
  - ..You can do whatever you want!
  - It is faster to run native code than scripts
- **If you're using ready-made startup scripts**
  - Eliminate unused stuff ("set –x")
  - Run multiple scripts in parallel..if possible

Userspace - upstart

- Replaces normal init
- Allows parallel execution of scripts and binaries
- Event driven – control the flow of execution
  - Jobs abstraction
- Events arrive explicitly, on start and stop of programs and specific conditions
- Compatible with SystemV scripts.

Application Prelink

- A good portion of application initialization time is spent resolving symbols to dynamic libraries
- Using Prelinking you can cut off a significant portion of application startup time
- Tries to assign a preferred address space to each library used by an application – ahead of time
- Only preferred. If unable, works as normal linking

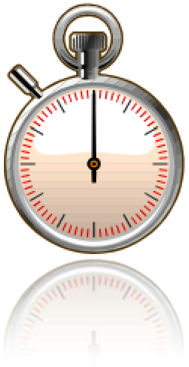
Agenda

- **Why Fastboot?**
- **The Boot Sequence**
- **Optimizations**
  - Bootloader
  - Kernel
  - Middleware & Applications
- **Customer Example – One Second Boot**

An Extreme Customer Example

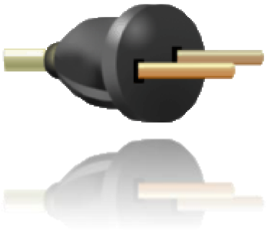
- Due to contractual reasons the customer cannot be named
- The use case was an Automotive dashboard application

Design Challenges



System must provide visual feedback by displaying critical real-time data in less then 1 second from power-on

Can not use resume/suspend due to very limited power budget and temperature range - no battery to backup RAM



Must be resilient to power loss at any time to prevent data corruption

The One Second Boot Demonstration

Also available at
http://www.youtube.com/montavistasoftware

The Solution

- **These magicians will reveal their secrets**
  - Well most of them anyway…
    - Get the most out of the hardware – use DMA to transfer the kernel image from NOR into RAM, 30% faster!
    - Parallelization is king! Use DMA to populate initramfs while kernel is still booting: 50% faster!
    - Get initial splash screen displayed quicker by copying image into framebuffer by DMA, 30% faster

Careful tuning of software stack from ground up starting from bootloader

Use highly optimized kernel and keep it uncompressed in NOR flash

Load only required drivers and do it as fast as possible

Resources

- **Arjan Van de Ven’s talk at LPC**
  - “LPC: Booting Linux in 5 Seconds”
    http://lwn.net/Articles/299483/
- **New fastboot git tree:**
  - "What's in fastboot.git for 2.6.28“
    http://lwn.net/Articles/299591/
- **Oprofile – Systemwide Stat. Profiling for Linux**
- **elinux wiki – Boot Time development portal**
  - http://elinux.org/Boot_Time
  - http://elinux.org/Tims_Fastboot_Tools
- **Tim Bird's OLS 2004 presentation**
  - Methods to Improve Bootup Time in Linux
  - http://kernel.org/doc/ols/2004/ols2004v1-pages-79-88.pdf
- **Arjan van de Ven, Linux Plumbers Conference Booting Linux in 5 Seconds (x86/Desktop focused) (Sept 18, 2008)**
  - http://lwn.net/Articles/299483/

- [Dean] Open Systems Media will put a contact slide here while we do the Q&A