Kernel- und Treiberprogrammierung mit dem Linux-Kernel – Folge 64

Kern-Technik

Der Boot Tracer und Bootchart helfen dabei, den Bootvorgang detailliert zu analysieren. Schon ein paar Maßnahmen später ist das Linux-System spürbar schneller betriebsbereit. *Jürgen Quade*, *Eva-Katharina Kunst*

Die heute gängige Bootzeit von ein bis zwei Minuten für ein Desktop-Linux ist nicht gerade beeindruckend. Für den Einsatz im Embedded-Bereich wäre sie geradezu undenkbar. Eine Digitalkamera, die 60 Sekunden zum Hochfahren benötigt, findet nicht viele Käufer, ebenso ein Auto, das nach dem "Keyless Access" zwei Minuten mit der Meldung "Fahrzeug-Informationssystem - Booting" für Stillstand sorgt.

Dabei geht es schneller, wie diverse im Internet publizierte Erfolgsgeschichten belegen. Dort wurden mit überschaubarem Aufwand Bootzeiten im einstelligen Sekundenbereich und teilweise sogar darunter erreicht. So bootet etwa das unter [1] gezeigte Embedded-System auf einem Beagleboard in 630 Millisekunden von Power-on bis in eine Shell.

Für einen möglichst schnellen Bootvorgang gilt es, die Systemkomponenten in Handarbeit anzupassen. Davor ist außerdem eine zeitliche Analyse des Bootvorgangs erforderlich, der sich beim so genannten Cold Boot in die vier Phasen Bootstrap, Bootloader, Kernel und Userland gliedert (siehe Abbildung 1).

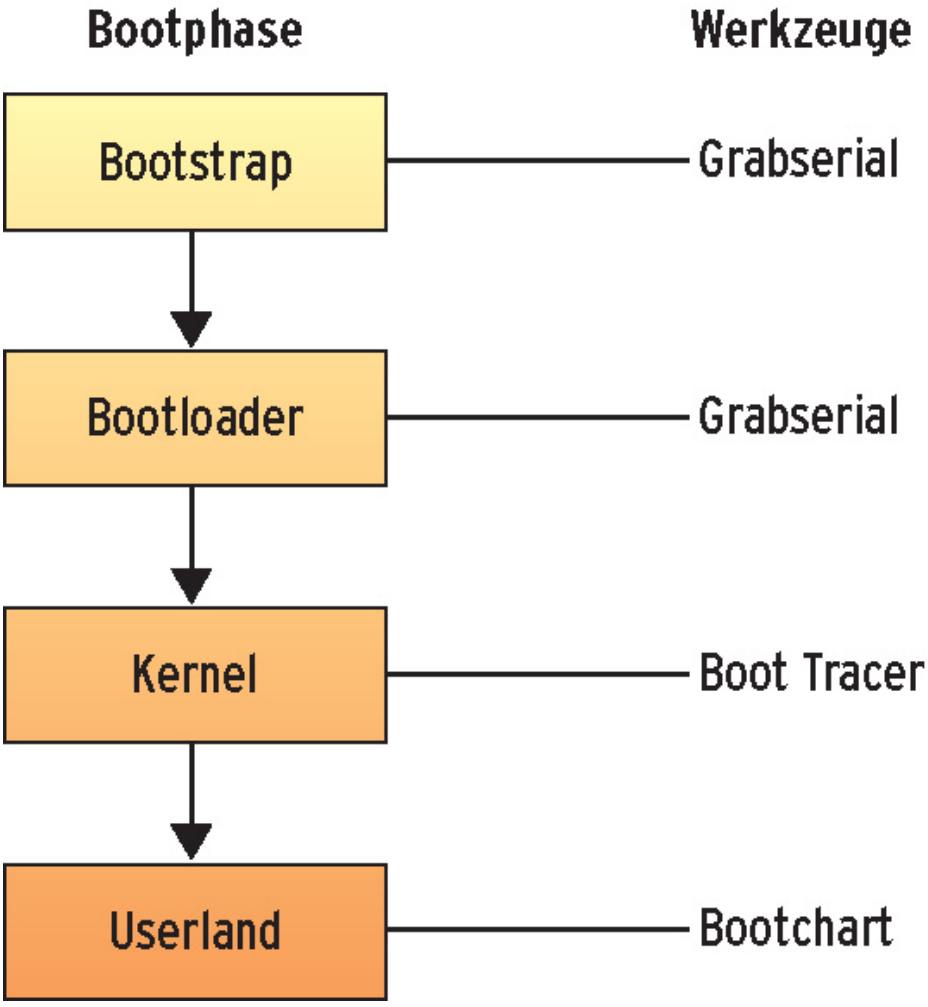

Abbildung 1: Je nach Bootphase stehen zur Analyse unterschiedliche Werkzeuge bereit.

Die Phase Bootstrap entspricht dabei dem Bios beziehungsweise der Abarbeitung des Codes, der in einem Festspeicher (ROM) untergebracht ist. Der typischerweise einfache Bootstrap-Code lädt am Ende den Bootloader, der wiederum für das flexible Laden eines Root-Dateisystems und des Betriebssystems zuständig ist.

Seriell beobachtet

Für die zeitliche Analyse der Bootstrap- und der Bootloader-Phase empfehlen die Kernelhacker das Werkzeug Grabserial [2], das sich allerdings nur einsetzen lässt, wenn das zu untersuchende System über eine serielle Schnittstelle verfügt. Grabserial läuft dabei nicht auf dem zu untersuchenden System selbst, sondern auf einem eigenen Host, der die über die serielle Schnittstelle ausgegebenen Bootnachrichten empfängt.

Die Aufgabe ist simpel: Zu jeder Nachricht fügt die Software einen genauen Zeitstempel hinzu. Die Auswertung der zeitbehafteten Nachrichten von Bootstrap und Bootloader bleibt aber Handarbeit für den Entwickler. Auch in der Phase des Kernelboots lässt sich Grabserial einsetzen. Etwas komfortabler geht es aber mit Boot Tracer zu. Das Werkzeug ist seit Version 2.6.28 Teil des Linux-Kernels in der so genannten Tracing-Infrastruktur. Sobald Boot Tracer durch die Bootoption

```
initcall_debug printk.time=1
```

aktiviert ist, protokolliert es die Start- und Endzeitpunkte der Kernel-Initialisierungsfunktionen, der so genannten Initcalls (siehe Abbildung 2). Es registriert die Zeitdauer ebenso wie Erfolg oder Misserfolg einer Init-Funktion. Der Kernelhacker Arjan van de Ven hat darüber hinaus mit »bootgraph.pl« ein Perl-Skript geschrieben, das die vom Boot Tracer generierten Informationen grafisch aufbereitet. Torvalds legt dieses Skript seinem Kernel-Quellcode im »scripts«-Verzeichnis bei.

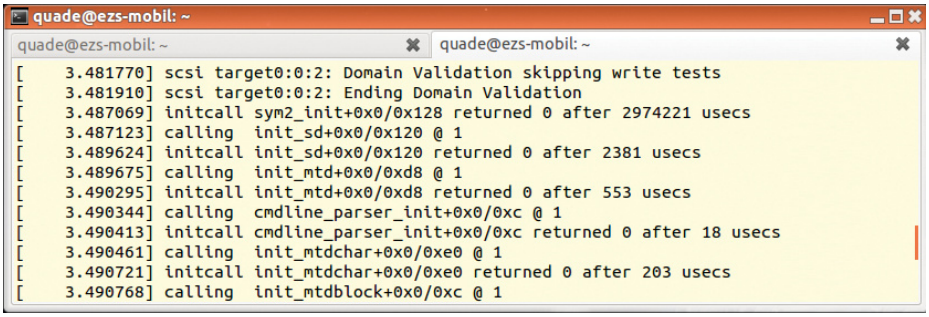

Abbildung 2: Der Boot Tracer protokolliert Zeitverhalten und Returncodes der Initialisierungsfunktionen.

Da Distributionen wie beispielsweise Ubuntu die Tracing-Infrastruktur im Kernel unterstützen, kann der Anwender den Bootprozess mit wenig Aufwand selbst analysieren (siehe Kasten "Boot Tracer im Einsatz"). Das Augenmerk ist in der generierten Grafik auf die lang gezogenen Felder zu richten. Abbildung 4 beispielsweise zeigt die Boot-Tracer-Ausgaben für ein System, das mit den Default-Einstellungen von Buildroot [3] für ARM entstand. Demnach dauert das Booten des Kernels bis zum Mounten des Root-Filesystems unter dem Emulator Qemu 3880 Millisekunden.

Boot Tracer im Einsatz

Ein Reboot reicht aus, um den Boot Tracer unter Ubuntu 12.04 selbst auszuprobieren. Dazu ist es zunächst erforderlich, in das Bootmenü von Grub zu gelangen. Falls beim Reboot der Bootmanager Grub nicht selbstständig erscheint, lässt er sich durch Drücken der Umschalt-Taste aufrufen. Im Menü ermöglicht es die Taste `[E]`, den aktuellen Eintrag zu editieren. Daraufhin erscheint der ausgewählte Eintrag im Detail (siehe Abbildung 3).

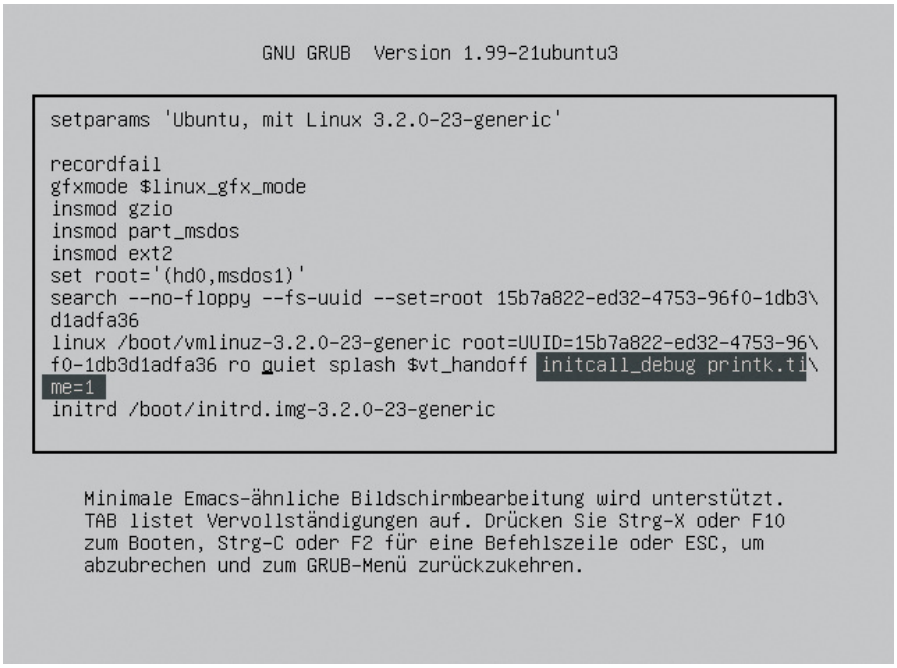

Abbildung 3: Mit den geeigneten Kommandozeilen-Optionen gestartet protokolliert der Boot Tracer den Kernelboot.

Der Cursor gehört in die Zeile, die mit »linux« beginnt. Dort sind die beiden Kerneloptionen »initcall_debug printk.time=1« anzufügen. Dabei gibt es eine kleine Herausforderung: Zum Bootzeitpunkt ist das Tastaturlayout auf Amerikanisch eingestellt. Für den Unterstrich ist ein `[?]` zu tippen, für das Gleichheitszeichen ein `[']`. Die Eingabe von `[Strg]+[X]` bootet den Kernel mit den neuen Optionen.

Nach dem Booten speichert man die Protokollinformationen in einer Datei. Das Skript »bootgraph.pl« bereitet deren Inhalt grafisch auf:

```
dmesg >/tmp/boot.log
cat /tmp/boot.log | U
```

Je nach den installierten Kernel-Headerdateien sind die angegebenen Pfade eventuell noch anzupassen. Mit einem Bildbetrachter wie etwa »eog« lässt sich die erzeugte Grafik »/tmp/boot.svg« betrachten. Da sie sehr viele Informationen enthält, sind in der Standardansicht zunächst nur die großen Blöcke sichtbar, das Zoomen der interessanten Ausschnitte offenbart die Details.

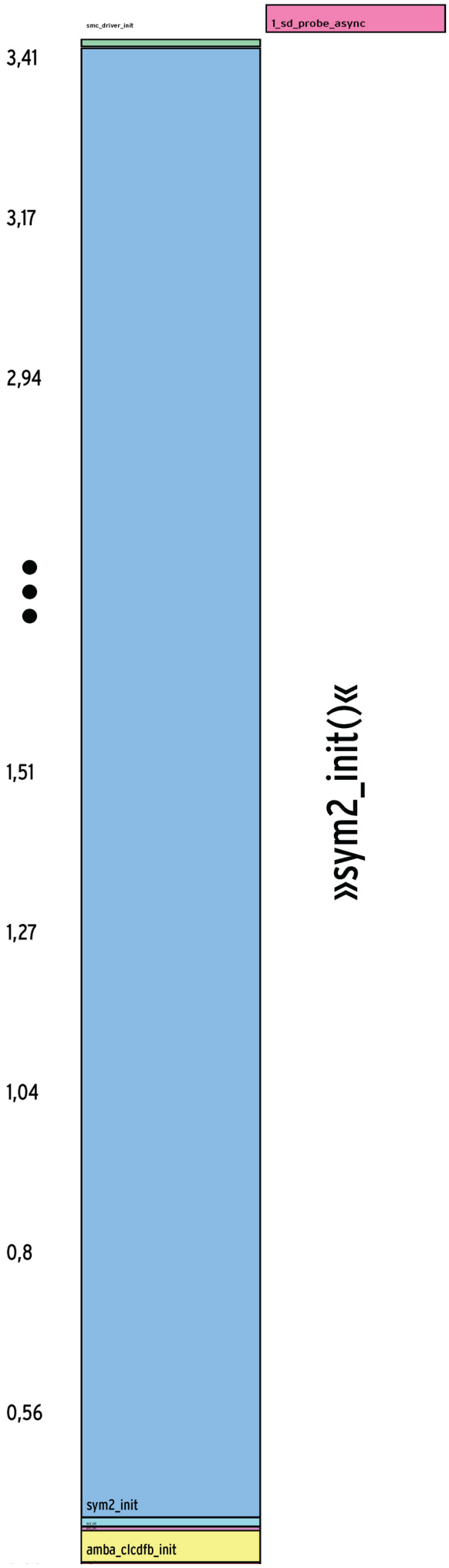
smc_driver_init
1_sd_probe_async
3,41
3,17
2,94
1,51
1,27
1,04
0,8
0,56
»sym2_init()«
sym2_init
amba_clcdfb_init

Abbildung 4: In der Boot-Tracer-Grafik ist deutlich der lange blaue Bereich erkennbar, der für die lange Bootzeit verantwortlich ist.

Eingebaute Wartezeit

Auffällig ist der etwa 3 Sekunden lange blaue Bereich, den die Funktion »sym2_init()« verursacht. Eine Quellcode-Recherche zeigt, dass diese Funktion zum SCSI-Treiber »sym53c8xx« gehört. Im Rahmen der Funktion wartet der Kernel über einen 3-Sekunden-Timeout auf SCSI-Geräte. Dieses Verhalten lässt sich allerdings per Bootoption abstellen. Startet man den gleichen Kernel mit der Bootoption »sym53c8xx.settle=0«, ergibt sich der in Abbildung 5 dargestellt Boot Trace. Bis zum Mounten der Rootpartition benötigt das System dann nur noch etwa 900 Millisekunden.

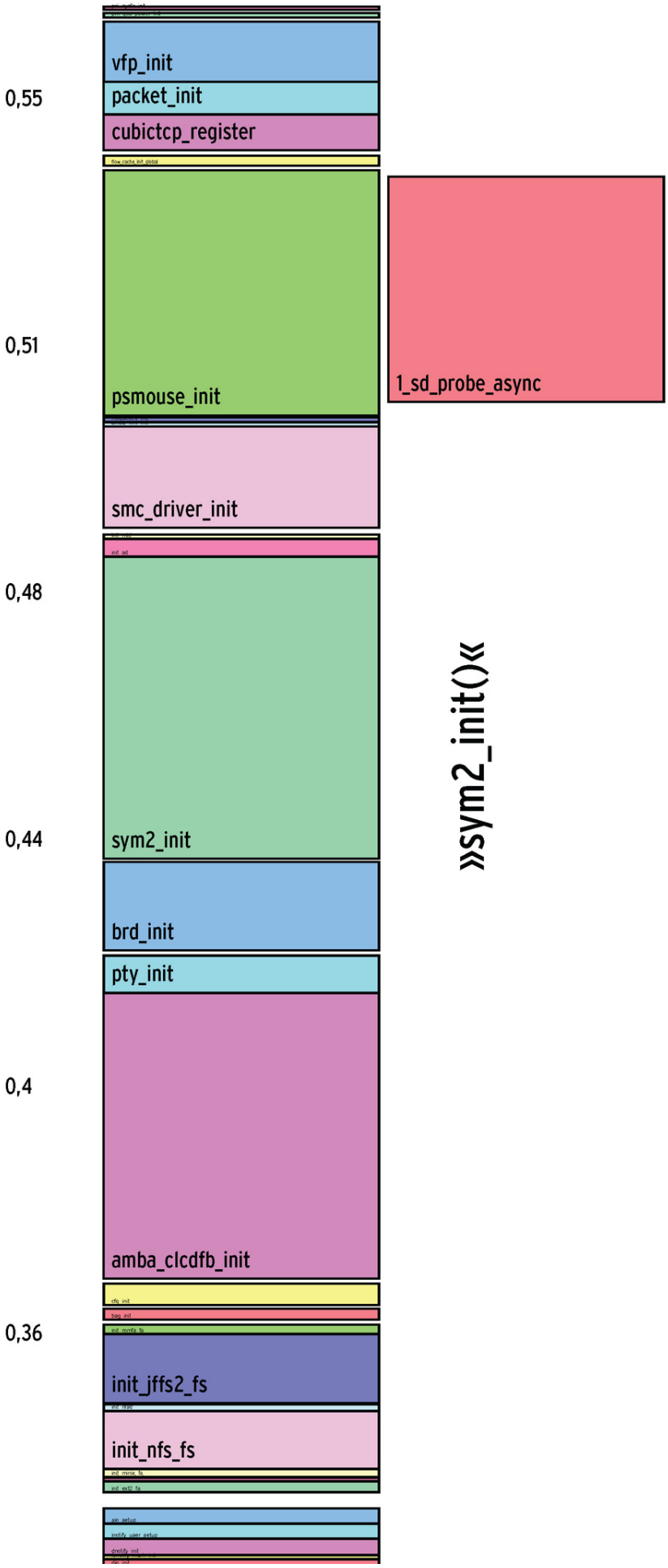
0,55
0,51
0,48
0,44
0,4
0,36
vfp_init
packet_init
cubictcp_register
psmouse_init
1_sd_probe_async
smc_driver_init
sym2_init
»sym2_init()«
brd_init
pty_init
amba_clcdfb_init
init_jffs2_fs
init_nfs_fs

Abbildung 5: Mit der passenden Bootoption gestartet zeigt das Linux-System in der Boot-Tracer-Auswertung eine deutlich verkürzte Startzeit.

Über die detaillierte Auswertung der Boot-Tracer-Informationen hinaus schlagen die Kernelexperten eine Reihe weiterer Maßnahmen vor ([http://4], [5], [6], [7]). Einige empfehlen, den Konsolen-Output während des Bootens zu unterbinden. Auch diese Maßnahme lässt sich per Bootoption umsetzen (Schlüsselwort »quiet«). Außerdem sollte man Parameter, die der Kernel beim Booten jedes Mal aufs Neue bestimmt, vorgeben. Ein prominentes Beispiel ist das Kalibrieren der Delay-Loop. Der vom Kernel durch Messung ermittelte Wert taucht in den Syslog-Meldungen auf. Zum Abkürzen des Vorgangs übergibt ihn der Admin gleich als Bootoption »lpj=*Wert*«.

Ein weiterer Tipp betrifft die Kernelkonfiguration selbst. Es versteht sich von selbst, dass ein schlanker Kernel schneller bootet als ein aufgeblasener. Konsequenterweise sollte man nur jene Komponenten einkompilieren, die Hardware-technisch verbaut beziehungsweise Software-technisch nötig sind. Das betrifft nicht nur Treiber, sondern insbesondere auch Netzwerkprotokolle, Dateisysteme und Debugoptionen. Eine weitere Technik wird als Deferred Loading bezeichnet. Funktionalitäten, die das System nicht direkt zum Start benötigt, lädt es erst nach dem Booten.

Die Experten erwähnen als weitere Möglichkeit zur Zeitersparnis, die Anzahl der Pseudo-Terminals (PTY) zu reduzieren, was ebenfalls per Bootoption ohne Neukompilieren funktioniert. Ein zusätzlicher Tipp ist das Abschalten der zuweilen noch aktiven Autokonfiguration des Internetprotokolls im Kernel (»ip-auto-config«).

Minimal-Bootstrap

Damit nicht genug. Die Systemarchitekten von Linux hinterfragen zur Optimierung der Startzeit den ganzen Bootprozess. Dabei zeigt sich, dass beim Kalt-Boot auch der Bootloader zusammen mit dem Laden von Kernel und Root-Filesystem viel Zeit verschlingt. Die Embedded-Experten stellen zur Diskussion, den Kernel direkt mit einem einfachen Bootstrap in den Speicher zu laden und einen mit reichlicher Funktionalität versehenen Bootloader komplett wegzulassen. Dazu muss man die Bootoptionen allerdings fest in den Kernel integrieren. Das ist bei der Kernelkonfiguration über den Parameter »CONFIG_CMDLINE« möglich.

Das Laden des Kernels und des Root-Dateisystems durch den Bootstrap lässt sich ebenfalls optimieren. Zu diesem Zweck kann der Admin den Transfer per DMA vornehmen oder auch mit LZO einen schnelleren Dekompressions-Algorithmus verwenden als standardmäßig vorgesehen.

Der letzte Punkt, bevor die Optimierung für das Userland beginnt, betrifft die Konfektion des Root-Filesystems. Abhängig von der Größe der Rootpartition und des verwendeten Filesystems benötigt das Mount-Kommando unterschiedlich lange Zeit. Auch ein Read-only-Mounten kann Zeit einsparen. Bei der Auswahl eines kurz bootenden Filesystems kann der Benchmark unter [8] helfen.

Schneller User

Für die Analyse der letzten Bootphase, also das Hochfahren des Userland, haben die Linux-Entwickler das Werkzeug Bootchart geschaffen. Für Ubuntu gibt es ein vorgefertigtes Paket, das sich mit dem Kommando »apt-get install bootchart« installieren lässt. Bereits nach dem nächsten Reboot stehen die Daten zur Verfügung und können analysiert werden. Da Ubuntu allerdings nach dem ersten Reboot noch eine Systemoptimierung vornimmt, ist es sinnvoll, frühestens die Daten des zweiten Reboots

auszuwerten. Ist die Auswertung abgeschlossen, entfernt der Befehl »apt-get remove bootchart« das Tool wieder.

Kern von Bootchart ist ein Hintergrundprozess namens »collector«, der in kurzem Abstand, beispielsweise alle 40 Millisekunden, die Taskliste, statistische Informationen über Tasks und Interrupts (»/proc/stat«) sowie statistische Informationen über den Disk-I/O (»/proc/diskstats«) protokolliert. Die gesammelten Informationen packt das Werkzeug in ein Archiv und legt es im Verzeichnis »/var/log/bootchart« ab.

Es bereitet die Daten auch grafisch auf. Die Grafik (Abbildung 6) findet sich ebenfalls im Verzeichnis »/var/log/bootchart« und wartet auf die Analyse. Neben allgemeinen Informationen ist in Blau die CPU-Auslastung und in Rosa die Disk-Aktivität während des Bootvorgangs dargestellt. Stellen, an denen die CPU keine Aktivität zeigt, sind gute Kandidaten für eine Optimierung.

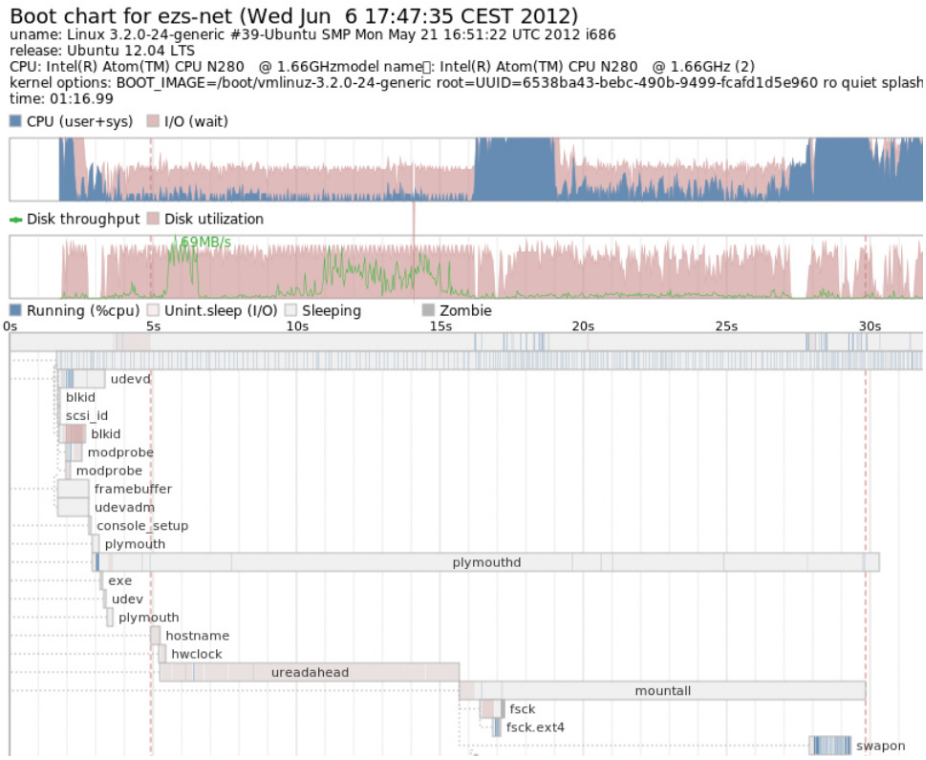

Abbildung 6: Bootchart malt detailliert auf, welche Task zu welchen Zeitpunkten aktiv ist.

Im weiteren Verlauf ist jede Task in der Grafik entlang der Zeitachse angeordnet. Dabei kennzeichnet Bootchart die Prozesszustände (aktiv, schlafend, Zombie) und Datentransfers. Abbildung 6 zeigt einen kleinen Ausschnitt aus dem Bootchart, das den Boot eines aktuellen Ubuntu-Systems auf einem Netbook abbildet. Die Collector-Task ist ebenfalls aufgeführt. Dabei ist deutlich die regelmäßige Aktivität zu sehen und auch, dass der Collector, durch Init aktiviert, frühzeitig im Bootprozess startet.

Eine für die Bootzeit-Optimierung wichtige Task ist die ebenfalls sichtbare und frühzeitig gestartet Task »ureadahead« [9]. Sie spart gut 30 Sekunden Bootzeit ein, indem sie vorausschauend möglichst viele Daten, die im weiteren Verlauf beim Hochfahren benötigt werden, in einem Rutsch in den Hauptspeicher transferiert. Welche Daten einzulesen sind, hat »ureadahead« bei einem früheren Bootvorgang durch Beobachtung der Plattenzugriffe gelernt. Dieses vorausschauende Lesen gibt es nicht nur für Ubuntu-Systeme, es steht auch im schlanken Universal-Binary Busybox für Embedded Linux zur Verfügung.

Als Alternative zu »ureadahead« wird auch ein Werkzeug namens »e4rat« gehandelt, das die zum Booten benötigten Daten auf der Festplatte zusätzlich so umsortiert, dass sie in

einer sinnvollen Reihenfolge hintereinanderliegen [10]. Tests anlässlich dieses Artikels konnten gegenüber »ureadahead« aber keine signifikanten Verbesserungen messen.

Langsame Automatik

Für die Optimierung der Userland-Bootzeit schlagen die Experten einen ganzen Reigen unterschiedlicher Maßnahmen vor. Zunächst sollte der Admin alle nicht benötigten Dienste entfernen und benötigte – falls möglich – nach dem eigentlichen Booten starten. Sehr viel Zeit spart, wer Gerätedateien nicht automatisch per Udev, sondern statisch anlegt. Erfolg versprechend ist es auch, auf den Init-Mechanismus zu verzichten und lieber die einzelnen Dienste mit einem eigenen Startup-Skript zu starten. Für ein Desktop-Ubuntu ist das sicherlich keine geeignete Maßnahme, für ein Embedded-System aber sehr wohl.

Für dieses Umfeld sind auch die weiteren Tipps gedacht, nämlich optimierte C-Bibliotheken zu verwenden, Applikationen statisch zu linken und unnötige Informationen mittels »strip« zu entfernen. Wer noch mehr Zeit rauskitzeln möchte, prüft schließlich die Applikationen selbst und versucht per Profiling deren Laufzeitschwächen aufzudecken.

Die im Internet abrufbaren Erfolgsmeldungen zeigen einheitlich, dass eine signifikante Reduktion der Bootzeit mit ein, zwei, vielleicht auch drei Tagen Arbeit zu erreichen ist. Danach mag immer noch Potenzial für weitere Verkürzungen bestehen, der Aufwand steigt aber unverhältnismäßig. *(mhu)*

Infos

1. Make Linux Software, "Super Fast Boot of Embedded Linux": [http://www.makelinux.com/emb/fastboot/omap]
2. Grabserial: [http://elinux.org/Grabserial]
3. Quade, Kunst, "Kern-Technik", Folge 62: Linux-Magazin 05/12, S. 84
4. Michael Opdenacker, "Cheap Linux boot time reduction techniques": [http://free-electrons.com/pub/conferences/2011/genivi/boot-time.pdf]
5. Chris Simmonds, "Reducing boot time in Linux devices": [http://www.embedded-linux.co.uk/downloads/WE-2.2Paper_Simmonds.pdf]
6. Embedded-Linux-Wiki, "Boot Time": [http://elinux.org/Boot_Time]
7. Sanjeev Premi, "Optimize Linux Boot Time": [http://processors.wiki.ti.com/index.php/Optimize_Linux_Boot_Time]
8. Free Electrons, "Flash Filesystem Benchmarks": [http://elinux.org/Flash_Filesystem_Benchmarks]
9. Scott James Remnant (keybuk), "All about ureadahead": [http://ubuntuforums.org/showthread.php?t=1434502]
10. Lifehacker, "E4rat Seriously Cuts Down on Linux Boot Time With a Few Simple Commands": [http://lifehacker.com/5790311/e4rat-cuts-your-linux-pcs-boot-time-in-half-with-a-few-simple-commands]

Die Autoren

Eva-Katharina Kunst, Journalistin, und Jürgen Quade, Professor an der Hochschule Niederrhein, sind seit den Anfängen von Linux Fans von Open Source. In der Zwischenzeit ist die dritte Auflage ihres Buches "Linux Treiber entwickeln" erschienen.