Linux Kernel

Egretzberger Dominik, Porcic Alin

18. Januar 2015

Inhaltsverzeichnis

1 Allgemeines

Linux ist kein Betriebssystem, sondern ein monolithischer Kernel, der sehr oft in Kombination mit GNU-Programmen als sogennante Distributionen verfügbar ist.

- läuft auf 27 verschiedenen Architekturen
- in C und Assembler geschrieben
- hält sich an den POSIX-Standard
- Open Source (GPL)

2 Was ist ein Kernel?

Ein Kernel ist der zentrale Bestandteil eines Betriebssystems und legt die Prozess- und Datenorganisation fest. Zudem ist er für die Trennung der Benutzerprogramme und der Hardware zuständig und für das Managment der Treiber. Es gibt verschiedene Typen von Kernel, die sich im wesentlichen nur davon unterscheiden, welche Betriebssystemkomponten im Kernel integriert sind und welche im Userspace.

- **monolithischer Kernel**: Dieser Typ von Kernel implementiert ein virtuelles Filesystem (VFS), managt die Dateisystem-Zugriffe und Gerätetreiber.
- **Mikrokernel**: Diese Kernel sind sehr minimalistisch gehalten und implementieren nur die nötigsten Funktionen, die ein Kernels braucht. Treiber, Anwendungs-IPC (Inter-Process-Communication) und Dateisystem müssen von Benutzerprogrammen impementiert werden.
- **Hybridkernel**: Ist ein Mix aus monolithischen Kernel und Mikrokernel. Gerätetreiber und Anwendungs-IPC sind im Kern vorhanden, Dateisystem läuft im Userspace.

Bei allen Kernel müssen folgende Dinge implemeniert sein, da es sich sonst um keinen Kernel mehr handelt:

- **Scheduler**: Der Scheduler entscheidet zu welcher Zeit welcher Prozess wie lange die CPU belegen darf.
- **Basic IPC**: In irgendeiner Form muss den Benutzerprogrammen eine Möglichkeit geboten werden miteinander zu kommunizieren.
- **Virtual Memory**: Der Kernel proiziert einen sogennanten 'virtuellen Speicherplatz' auf realen Speicherplatz. Bei Linux wird der virtueller Speicher jedes Prozesses in Pages (=Seiten) aufgeteilt und diese werden dann auf realen Speichermedien proiziert.

3 Geschichtliche Hintergründe

1970 wurde Unix veröffentlicht - ein erstmal in Assembler und später in C geschriebenes Betriebssystem. Unix wurde an Universitäten für Lernzwecke verteilt und hatte bis dahin keine Lizenz definiert. Jedoch änderte sich das, da das Unternehmen 'Bell Laboritories' (heute AT&T) profit erwirtschaften wollten. Das führte dazu, das Unix sehr teuer wurde und nur wohlhabende Unternehmen es sich leisten konnten. Daher gründet Richard Stallman die GNU-Foundation im Jahre 1986 mit dem Ziel ein alternatives Betriebssystem zu schaffen, damit private Personen auch kostenlos Zugang zu einem ordentlichen Betriebssystem haben können.
Nach einigen Jahre war das Betriebssystem so gut feritg, aber es gab ein Problem: der GNU Hurd Kernel. Der Hurd Kernel arbeitete mit vielen Threads, daher war das Debuggen des Kernels sehr schwierig und zeitaufwendig. Zufällig arbeitet Linus Torvalds in Finnland an einer Universität an einem Hobbyprojekt: er wollte am Betriebssystem Minix seinen eigenen Kernel schreiben, um den Computer besser zu verstehen.
Als Linux 1992 funktionstüchtig war, veröffentlichte Linus den Kern unter der GPL (Gnu-Public-Licence). Seither wird weltweit daran gearbeitet und Linus selbst ist auch noch dabei.

4 Linux auf Embedded Systems

Linux auf Embedded Systems ist grundsätzlich kein Problem, da Linux sehr gut skalierbar ist und mit den richtigen Anpassungen sowohl auf Supercomputer als auch auf Mikrocontroller läuft. Viele Programme und Bibliotheken im Userspace gibt es auch als abgespeckte Versionen, die speziell für Embedded Systems und resourcenarme Systeme ausgelegt sind (EGLIBC oder uClibc statt glibc), oder es gibt ressourcenarme Alternativen. Einige Anpassungen von Linux sind nötig gewesen, um Linux auf Embedded Systems lauffähig zu machen.

5 Linux Bootvorgang

5.1 Bootvorgang PC

Der Bootvorgang am Computer gliedert sich in den POST (wird vom BIOS ausgeführt) und den Bootloader, der dann das eigentliche System startet. Das BIOS führt den POST, einen Selbsttest, durch und startet dann einen Bootloader, den er auf einem startfähigen Gerät findet (Festplatte, USB-Stick, etc.). Der Bootloader kann über das BIOS mit den Geräten kommunizieren, denn es abstrahiert die unterschiedliche Hardware. Er lädt das Betriebssystem, dass dann die Kontrolle übernimmt und den Bootvorgang abschließt. Im Falle von Linux lädt der Bootloader, meist GRUB, den Kernel und eine “initial ramdisk” in den Arbeitsspeicher und startet dann den Kernel. Der Kernel benutzt den Inhalt der Ramdisk, um die Rootpartition einzuhängen.

Während der Bootphase kümmert sich das BIOS um die Initialisierung der Hardware, der Bootloader findet also bereits vorbereitete Hardware vor. Auch

kann der Bootloader über das BIOS mit der Hardware kommunizieren, ohne Treiber für jeden Controller und jede Festplatte zu benötigen. Das Betriebssystem kann über das BIOS die Kenndaten der Hardware, Speicheradressen und IRQs erfragen. Danach lädt es entsprechende Treiber für die Geräte.

5.2 Bootvorgang Embedded System

Der Bootvorgang bei Embedded Systems gliedert sich in eine Initialisierungsphase, die vom Bootloader übernommen wird, und in das Laden des Betriebssystems.

Der Bootloader befindet sich meistens in einem Flashbaustein, dessen Inhalt beim Einschalten des Prozessors geladen wird. Der Bootloader selbst findet keine initialisierte Hardware vor, er muss die komplette Hardware selbst vorbereiten und "aufräumen". Nach der Initialisierung kann der Bootloader ein Betriebssystem laden. Aufgrund der unterschiedlichen Hardware werden auch gerne Forks von generischen Bootloadern eingesetzt, die dann speziell auf die Hardware zugeschnitten sind. Dem System liegt jetzt zwar initialisierte Hardware vor, doch weil es kein BIOS gibt, weiß das Betriebssystem nichts über die Hardware.

In der Anfangszeit versuchten die Linux-Kernelentwickler so viele Embedded Systems wie möglich zu unterstützen. Durch das große Angebot and Hardware und tonnenweise Abwandlungen entstand schnell ein Chaos. Überall im Kern verteilt war Initialisierungslogik für die unterschiedlichsten Plattformen und Linus selbst drohte bald damit, keinen Code mehr für ARM Embedded Systems in den Kernel zu übernehmen, sollten die ARM Entwickler nicht bald eine Lösung für das Chaos haben.

Aufgrund dieser Drohung wurde eine Lösung entwickelt, der "Device Tree" für Linux. Der Device Tree ist eine baumförmige Datenstruktur, die Informationen über die Hardware beinhaltet (alle Informationen, die sonst das BIOS bereithalten würde), dieser wird kompiliert in einer Datei gespeichert, dem "device tree blob". Der Bootloader lädt nun nicht nur den Kernel und eine Ramdisk, sondern zusätzlich diese Datei. Der Linux-Kernel kann die Informationen aus dem Device Tree lesen (noch vor dem verwenden der Ramdisk), kennt nun die Hardware, die ihm zur Verfügung steht, und kann jetzt Treiber für die darin definierten Geräte laden. Danach unterscheidet sich der Bootvorgang nicht mehr von dem eines PCs.

Als Bootloader für ARM-Prozessoren kommt häufig "Das U-Boot" zum Einsatz, denn es ist äußerst flexibel und bietet viele Konfigurationsmöglichkeiten.