

Der Linux Kernel

Tilmann Bitterberg

<http://tibit.org/linux.html>

HfT-Stuttgart, den 15.06.2001

1 Einführung

1.1 Was ist Linux

- Ein Unix ähnliches Betriebssystem, das auf einer Vielzahl von Hardware Architekturen läuft.
- kostenlos aus dem Internet herunterzuladen <http://www.kernel.org>.
- besteht aus einem Kernel (*Linux*) und einem Betriebssystem (*GNU*)

1.2 Minix

- von Andy Tanenbaum für 8088er PCs (original IBM-PC) entwickeltes Lehrbetriebssystem.
- Microkernel Design.
- sehr restriktive Lizenz, Veränderungen durften nur in *patch* Form weitergegeben werden.
- Nicht kostenlos, ca. \$150 pro Lizenz.

1.3 Woher kommt Linux

- 1991 kaufte ein finnischer Informatik Student namens Linus Torvalds einen 386er PC.
- Unzufrieden mit DOS, wünschte er sich ein besseres Betriebssystem

- "Herumspielen" mit dem Lehrbetriebssystem Minix.
- Beschluß, ein besseres Minix als Minix zu schreiben.
- Durch die lange Geschichte von Unix, die Eleganz des grundlegenden Designs und die Verfügbarkeit einer Vielzahl an Software, beschloß Linus ein Unix-ähnliches Betriebssystem zu schreiben.

Am 5. Oktober 1991 schrieb Linus in der Newsgroup `comp.os.minix`:

'Do you pine for the nice days of Minix-1.1, when men were men and wrote their own device drivers? Are you without a nice project and just dying to cut your teeth on a OS you can try to modify for your needs? Are you finding it frustrating when everything works on Minix? No more all-nighters to get a nifty program working? Then this post might be just for you.

As I mentioned a month ago, I'm working on a free version of a Minix-lookalike for AT-386 computers. It has finally reached the stage where it's even usable (though may not be depending on what you want), and I am willing to put out the sources for wider distribution. It is just version 0.02...but I've successfully run `bash`, `gcc`, `gnu-make`, `gnu-sed`, `compress`, etc. under it.'

1.4 GNU

- GNU steht für GNU is not Unix
- Sammlung von freien Unix Programmen von der FSF.
- frei im Sinne von Redefreiheit, nicht Freibier.
- 1984 vom MIT Programmierer Richard Stallman entwickelt um eine freie, unix-ähnliche Arbeitsumgebung zu schaffen.
- Ursprünglich *hurd* als Kernel konzipiert, Linux war schneller.
- Bekannte Projekte aus dieser Sammlung: `emacs`, `gcc`, `bash`, usw.

1.5 GNU/Linux

- Benutzt GNU Software für die Arbeitsumgebung
- Benutzt Linux als Kernel
- Beides steht unter der *GNU Public License (GPL)*
- Die GPL erlaubt unbeschränkte Modifikation und Verteilung der Software.

1.6 Linux-Distributionen

- Bieten eine Sammlung von Programmen, die auf einander abgestimmt sind.
- Entwickeln Installationsanweisungen, -Skripte und Konfigurationstools.
- Man spart sich die Downloadkosten.
- Verlangen Geld für die Zusammenstellung und Support
- Bekannte Distributionen: RedHat, SuSE, GNU/Debian, Caldera

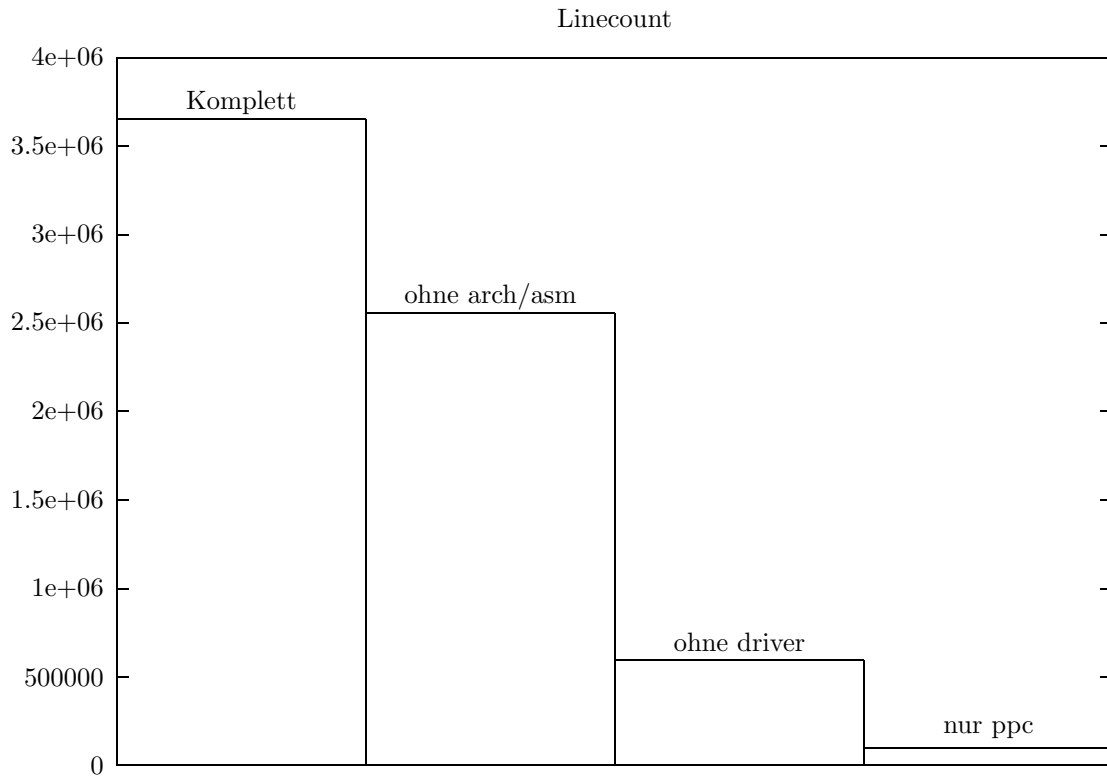
1.7 Wer braucht Linux?

Menschen, die

- ein modernes, zuverlässiges Betriebssystem haben wollen.
- wissen wollen, was hinter den Kulissen vorgeht.
- sich nicht mit teurer, fehlerbehafteter Software zufrieden geben.
- kostengünstige Webserver betreiben wollen.
- optimale Performance aus ihrem Rechner herausholen wollen.
- einfach Software entwickeln wollen.

1.8 Portabilität

- Linux 2.4 läuft auf 15 verschiedenen Hardware Plattformen
- durch Portierung auf viele Plattformen wurde eine saubere Schnittstelle geschaffen.
- Neue Architekturen hinzu zufügen ist relativ einfach.
- Gerätetreiber sind meist portabel geschrieben.



2 Einführung in den Linux Kernel

2.1 Download

- Zum Kernel lesen braucht man nur den Source.
- Gehe auf <http://www.kernel.org> oder direkt unter (ca. 24MB)
<ftp.kernel.org/pub/linux/kernel/v2.4/linux-2.4.5.tar.gz>
- entpacke das TAR Archiv. `tar xvzf linux-2.4.5.tar.gz`
- alles steht im Unterverzeichnis `linux/`

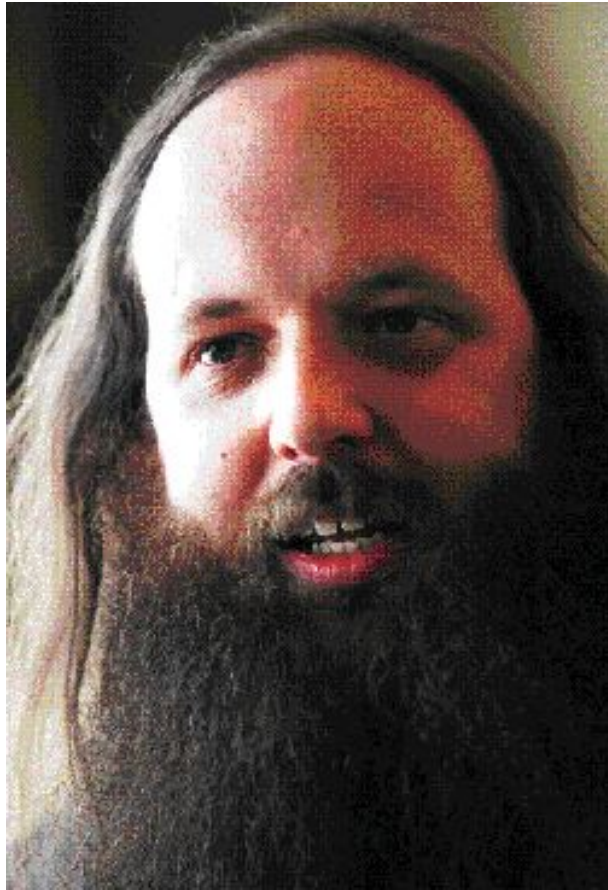
2.2 Wer programmiert Linux

- Tausende von Entwicklern, die über das Internet kommunizieren.
- Austausch über die *linux-kernel* Mailing Liste und IRC.
- Wöchentliche Zusammenfassung von *linux-kernel* unter <http://kt.zork.net>

- Linus Torvalds hat das letzte Wort was im Kernel ist.
- Alan Cox ist Linus rechte Hand.
- Jedes Subsystems/jeder Treiber hat seinen eigenen Maintainer
- Maintainer sammeln Code von anderen Leuten und schicken das an Linus
- Linus vertraut Maintainern.
- Alle Maintainer stehen in `linux/CREDITS`
- Linux Who-is-Who (mit Bildern) unter http://www.linux-mag.com/who/lmwho_1.html



Linus Torvalds

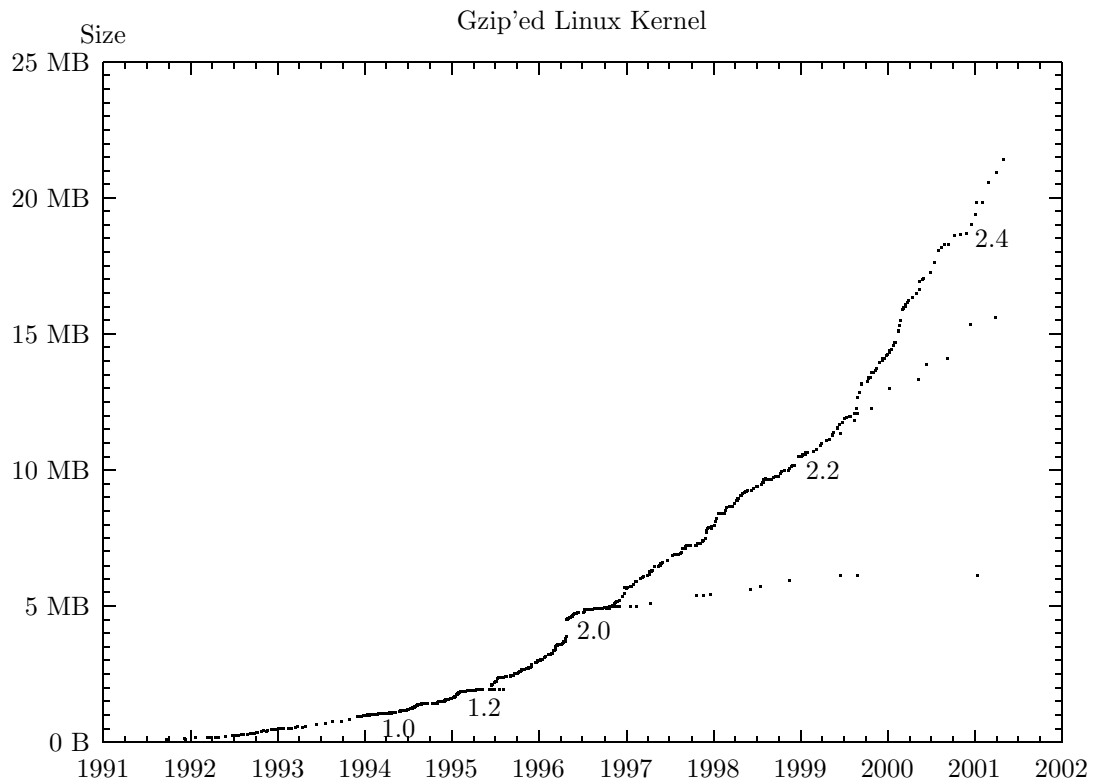


Alan Cox

2.3 Versionsnummern

- Zusammensetzung `VERSION.PATCHLEVEL.SUBLEVEL`
- aktuell: 2.4.5
- wenn `PATCHLEVEL` eine gerade Zahl, dann *stabile* Version
- wenn `PATCHLEVEL` eine ungerade Zahl, dann *Entwickler* Version
- aus jeder stabilen Version geht nach einiger Zeit eine Entwickler Version hervor; die nächste wird 2.5.x heißen.
- Alte Versionen haben durchaus ihre Berechtigung.
- Version 2.0 wurde bereits Juni 1996 freigegeben.

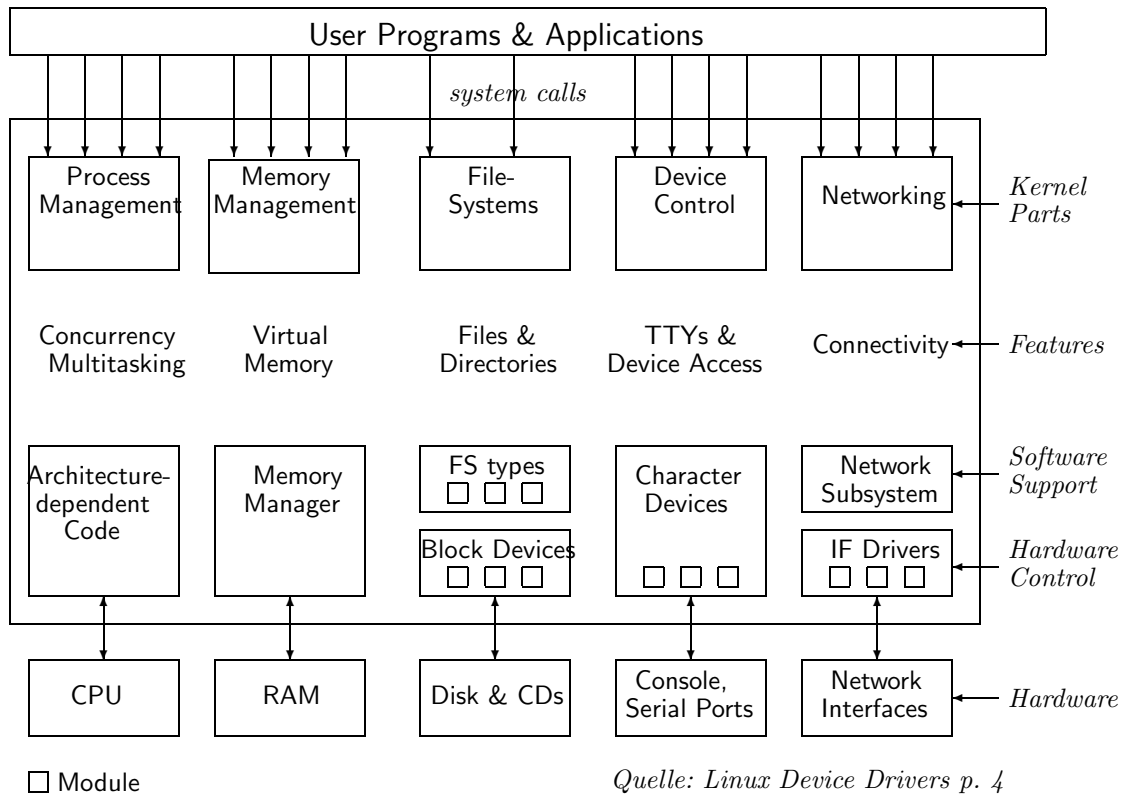
2.4 Zeitschiene



3 Struktur

3.1 Subsysteme

- Prozess Verwaltung
- Speicher Verwaltung
- Dateisysteme
- Geräte Kontrolle
- Vernetzung.



3.2 Physikalische Orientierung

- Aufteilung in logische Unterverzeichnisse; Beispiele:
- TV Karten
`linux/drivers/media/video/`
- IP-Stack
`linux/net/ipv4/`
- Netzwerk Karten
`linux/drivers/net/`
- Bootbarer Kernel
`linux/arch/i386/boot/bzImage`

linux/	
— arch/	Architekturspezifische Dateien
— Documentation/	Anwender Dokumentation
— drivers/	Gerätetreiber
— fs/	Dateisysteme
— include/	Öffentliche Header Dateien
— init/	Das main.c file des Kernels
— ipc/	Prozess Kommunikation
— kernel/	Core Kernel (eg. Scheduler)
— lib/	Nützliche Hilfsfunktionen
— mm/	Core Kernel Memory Management
— net/	Netzwerk Protokolle (Stacks)
— scripts/	Skripte, die das Leben leichter machen

3.3 linux/arch/*

- Aufspaltung von Linux in Rechner Architektur abhängige und unabhängige Teile.
- in arch/* steht Rechner Architektur abhängiges

linux/	
└─ arch/	
├─ alpha/	Frueher Digital, jetzt Compaq
├─ arm/	Emebedded, x86 aehnlich
├─ cris/	?
├─ i386/	Intel Pentium, AMD Athlon, ...
├─ ia64/	Intel Titanium 64Bit
├─ m68k/	Motorola M680000 (Amiga, Atari)
├─ mips/	SGI
├─ mips64/	SGI 64Bit
├─ parisc/	HP Risc Prozessoren
├─ ppc/	IBM/Apple/Motorola PowerPC
├─ s390/	IBM 31Bit s/390
├─ s390x/	IBM 64Bit s/390
├─ sh/	?
├─ sparc/	SUN Microsystems
└─ sparc64/	SUN Microsystems 64Bit

3.4 Beispiel i386

- Hauptsächlich Low-Level, Hardwarenahes
- Assembler

linux/	
└─ arch/	
└─ i386/	
├─ boot/	x86 Bootsektor, bootbarer Kernel
├─ kernel/	Low-Level Kernel, zB IRQ-Handling
├─ lib/	hochoptimierte Hilfsfunktionen
├─ math-emu/	Emulation des math. Coprozessors
└─ mm/	Low-Level Speichermanagent

3.5 Verzeichnisgrößen

linux (129368)	fs (9222)	nls (3204)
	include (18776)	linux (4268)
	net (6220)	
	drivers (64332)	net (15136)
		char (7164)
		scsi (11824)
		sound (4444)
		isdn (3544)
	arch (23932)	video (3920)
		usb (3044)
ppc (2916)		
Documentation (5220)	m68k (4340)	
	ia64 (2520)	

4 Algorithmen und Datenstrukturen

4.1 Allgemeines

- Alles möglichst simpel gehalten (dort wo es die Performance zuläßt)
- Viele Algorithmen "einfach" aus Lehrbüchern umgesetzt
- Einfache Listenstrukturen (einfach/doppelt verkettet, Queues)
- Fast alle Datenstrukturen dienen dazu, Asynchronität im Kernel bereitzustellen.

4.2 Scheduler

- Findet man in `linux/kernel/sched.c`

- Implementiert Round-Robin mit Prioritäten (goodness)
- Zeitscheibe ist ca. 20ms lang.

4.3 Virtueller Speicher

- LRU für swapping
- Balancing Zones
- Memory overcommitter (tut meistens)

4.4 Wait Queues

- Wenn ein Prozess etwas machen/haben will, was gerade nicht möglich ist, wird er in eine Warteschlange gesteckt.
- Will man selber Warteschlangen benutzen, kann man entweder eine schon vorhandene aus dem Kernel nehmen oder selber eine deklarieren mit `DECLARE_WAIT_QUEUE_HEAD()`, `DECLARE_WAIT_QUEUE()`
- Bekannte Warteschlangen ist z.B die `kswapd_wait`, in der der kernel swap daemon schläft, wenn es nicht zum swappen gibt.

4.5 Kernel Timers

- erweiterte Warteschlange.
- Benutzt man um eine Funktion zu einem späteren Zeitpunkt ausführen zu lassen.
- Das Datenelement wird automatisch aus der `timer_list` entfernt, wenn der timer abgelaufen ist.

4.6 Bottom Halves

- Innerhalb eines Interrupt handlers sollte nur relativ kurzer Code stehen.
- Dieser Code wird mit abgeschalteten Interrupts ausgeführt
- Wenn Code zu lang, gehen Datenpakete verloren.
- Bottom Halves bestehen aus zwei Teilen
- Kurzes Vorderteil, das während des Interrupts läuft
- Langes Hinterteil, das später ausgeführt wird.

4.7 Task Queues

- Erweiterung der Bottom Halves
- Mehr Flexibilität, dynamischer
- Benutzung mit `DECLARE_TASK_QUEUE()` und `queue_task()`

5 Spezielle Dateisysteme

5.1 `proc`

- beinhaltet Informationen über den Kernel und das laufende System.
- keine echten Dateien, werden vom Kernel beim lesen generiert.
- bietet einstellbare Kernel-Parameter zur Laufzeit.
- einfach programmierbar.

5.2 `devfs`

- statt statischen Geräte Einträgen in `/dev` wird bei `devfs` wirklich nur das angezeigt, was tatsächlich im System ist.
- logisch und physikalisch gegliedert.
- sorgt für Ordnung im `/dev` Verzeichnis.
- Früher die 1. Partition auf der 1. Festplatte im System hieß `/dev/hda1`, jetzt `/dev/discs/disc0/part1` oder `/dev/ide/host0/bus0/target0/lun0/part1`
- noch nicht alle Treiber wurden umgeschrieben.

5.3 `tmpfs`

- Schnittstelle zum RAM
- Kompatibel zum MIT `shm`
- wenn `tmpfs` gemounted ist, kann als ganz normales Filesystem darauf zugegriffen werden.

6 Module

- Module sind object files (.o), die nicht gelinkt sind
- Können dynamisch zur Laufzeit des Kernels hinzugeladen werden.
- Laufen dann im Kernspace, haben also unbeschränkten Zugriff auf alles.
- Laden mit insmod, entladen mit rmmod.
- Initialisierung mit `module_init()`, `module_exit()`

7 Beispiel hft-demo.c

```
/*
 * Demo proc entry
 *
 * Written by Tilmann Bitterberg
 */

#include <linux/module.h>
#include <linux/init.h> /* for module_init */
#include <linux/errno.h>
#include <linux/sched.h> /* for task struct */
#include <linux/proc_fs.h>

#include <asm/uaccess.h>

static ssize_t proc_demo_read(struct file * file, char * buf,
                             size_t count, loff_t *ppos);
static ssize_t proc_demo_write(struct file * file,
                              const char * buffer,
                              size_t count, loff_t *ppos);
struct file_operations proc_demo_operations = {
owner:          THIS_MODULE,
  read:         proc_demo_read,
  write:        proc_demo_write,
};

static ssize_t proc_demo_read(struct file * file, char * buf,
                             size_t count, loff_t *ppos)
{
int n = 0;
```

```

struct task_struct *p;
char buffer[512];

static int number=0;

number++;

n = sprintf (buffer ,
             "Ich wurde %d mal gelesen\nPids aktive: \n", number);
for_each_task(p)
    n += sprintf (buffer+n, "%d ", p->pid);
n += sprintf (buffer+n, "\n");

if (count < 0)
return -EINVAL;

if (*ppos >= strlen(buffer))
return 0;
if (n > strlen(buffer) - *ppos)
n = strlen(buffer) - *ppos;
if (n > count)
n = count;
copy_to_user(buf, buffer + *ppos, n);
*ppos += n;
return n;
}

static ssize_t proc_demo_write(struct file * file,
                              const char * buffer, size_t count, loff_t *ppos)
{
return 0;
}

MODULE_DESCRIPTION("Proc demo module");
MODULE_AUTHOR("Tilman Bitterberg");

static void cleanup_demoproc(void)
{
remove_proc_entry("hft-demo", NULL);
}

static int init_demoproc(void)
{
struct proc_dir_entry *entry;
entry = create_proc_entry("hft-demo", S_IRUGO|S_IWUSR, NULL);
if (entry) entry->proc_fops = &proc_demo_operations;
}

```

```
return 0;  
}
```

```
module_init(init_demoproc);  
module_exit(cleanup_demoproc);
```