

Linux Architektur

PD Dr. Reinhard Bündgen

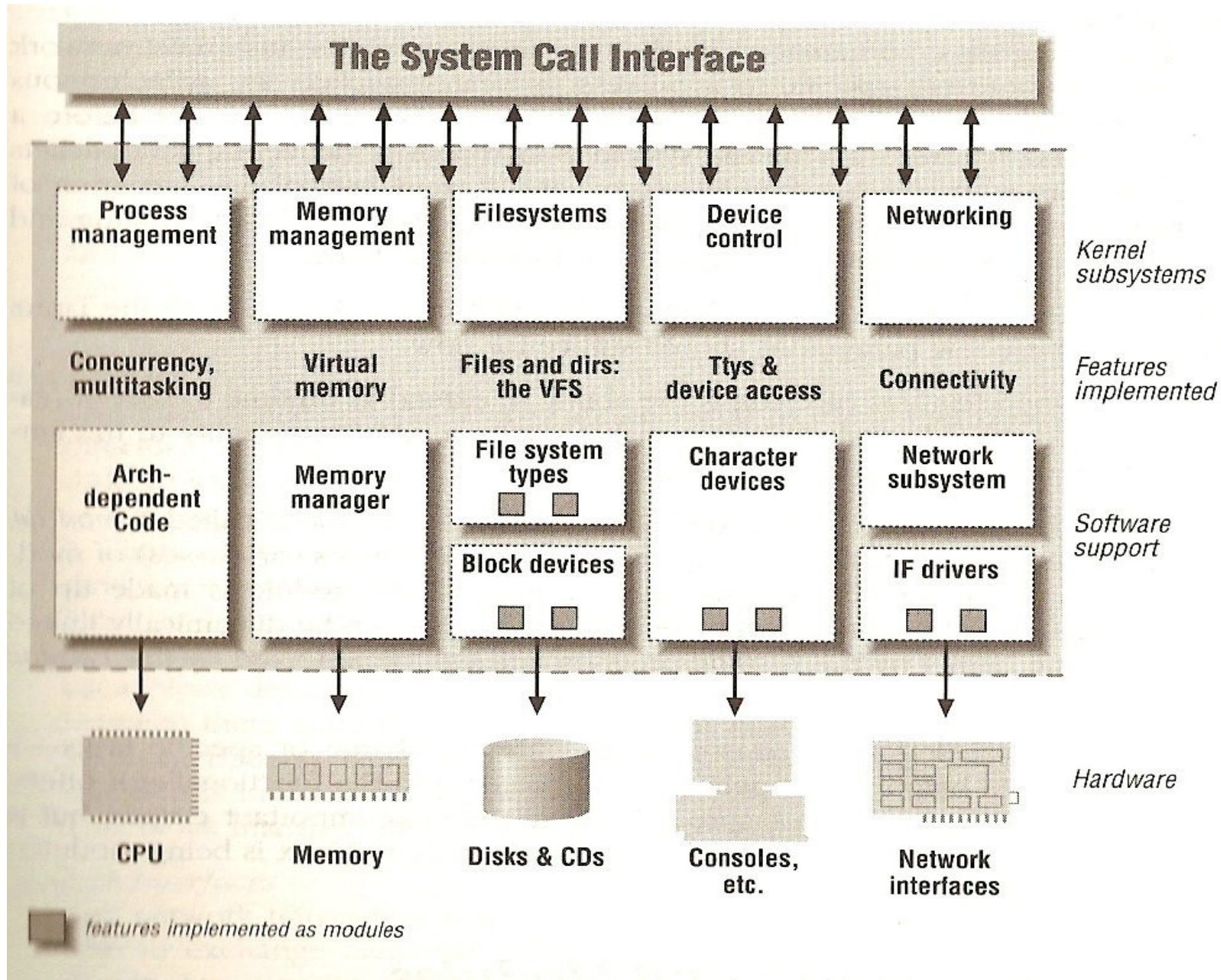
buendgen@de.ibm.com

Tel. 07031/16 1130

Linux Entwurfsziele

- Klarheit des Codes
- Kompatibilität
 - zu Standards (POSIX), vorhandener SW
- Portabilität
- Robustheit und Sicherheit
- Geschwindigkeit
 - C, Assembler, inline, Optimierung auf gcc

Linux Kern Struktur nach LDD



Kern Struktur (Forts.)

- Linux for S/390 System Struktur

Etwas Codestatistik

Linux 2.2

[nach Beck et al: Linux

Kernelprogrammierung (5. Aufl.)]

	C-Code ohne Headerdateien	Assembler- anweisungen
Gerätetreiber	800000	100
Netzwerk	146000	
VFS-Layer	17500	
25 Dateisysteme	108000	
Initialisierung	4000	3000
Koprozessor		3550
„Rest“	20000	

Linux 2.6.23.1

	Lines of code
arch/...*.c	1003729
...*.S	294801
block/...*.c	13125
crypto/...*.c	20819
drivers/...*.c	3291667
fs/...*.c	655860
init/...*.c	3026
include/...*.h	313487
ipc/...*.c	6748
kernel/...*.c	75461
mm/...*.c	29451
net/...*.c	43810
security/...*.c	23893
sound/...*.c	407626

Kern-Quellen

- <http://www.kernel.org>

```
buendgen@linux:~> cd linux-2.6.23.1
```

```
buendgen@linux:~/linux-2.6.18.1> ls
```

arch	crypto	include	kernel	mm	scripts
block	Documentation	init	lib	net	security
COPYING	drivers	ipc	MAINTAINERS	README	sound
CREDITS	fs	Kbuild	Makefile	REPORTING-BUGS	usr

```
buendgen@linux:~/linux-2.6.23.1>
```

Mikrokern vs. Makrokern

Mikro-Kern

- so klein wie möglich
- jedes Konzept läuft als eigener Prozess
- privilegierter Modus
- Kommunikation: Nachrichtenaustausch
- Abschottung unterschiedlicher Kernprozesse
- modular
- dynamischer Kernaufbau
- portabel
- sicher

Makro -Kern (Linux)

- gesamter Kern in gemeinsamen Adressraum
- Kommunikation
 - globale Variable
 - Funktionsaufruf
- schnell

Modularität im Linux-Kern

- LINUX-Kern Module sind Codebereiche, die dynamisch an den Kern gebunden bzw. vom Kern abgekoppelt werden können.
- Werkzeuge (modutils)
 - insmod: lade Module
 - rmmod: entferne Module
 - lsmod: liste geladene Module (vgl. /proc/modules)
 - depmod: bestimme Modulabhängigkeiten
 - modprobe: lade Modul dynamisch

Dynamisches Laden von Modulen

- Module können bei Bedarf dynamisch ge- und entladen werden.
 - usage count
- Standard Modullokation
 - `/lib/modules/<version>/kernel/{block,cdrom,fs,...}*.ko`
 - `/lib/modules/<version>/modules.dep`
- Modulspezifikation: in `/etc/modules.conf` oder in `/etc/modprobe.conf`
 - Pfadspez.: `path[misc]=...`
 - Modulename: `alias eth0 ne`
 - `alias block-major-35 xpram`
 - Parameter: `option xpram devs=4`
 - Pre/post Install & Remove sections

Module schreiben

- `#include <linux/module.h>`
- `init_module / module_init(fu_name)`
- `cleanup_module / module_exit(fu_name)`
- `module_param(Variable, Typ, Permissions)`
 - Weitere optionale Parameters für Externe Namen, Wiederholungen, Defaultwerte
- Beispiel: `linux/drivers/serial/serial_cs.c`

Hello World Module

```
/*
 * hello-3.c - Illustrating the __init, __initdata and __exit macros.
 */
#include <linux/module.h>      /* Needed by all modules */
#include <linux/kernel.h>      /* Needed for KERN_INFO */
#include <linux/init.h>        /* Needed for the macros */

static int hello3_data __initdata = 3;

static int __init hello_3_init(void)
{
    printk(KERN_INFO "Hello, world %d\n", hello3_data);
    return 0;
}

static void __exit hello_3_exit(void)
{
    printk(KERN_INFO "Goodbye, world 3\n");
}

module_init(hello_3_init);
module_exit(hello_3_exit);
```

Module:

Dokumentation/Lizenz/Sicherheit

- Dokumentation
 - `MODULE_AUTHOR („Name“)`
 - `MODULE_PARM_DESC (Name, Beschreibung)`
- Lizenz
 - `MODULE_LICENSE („GPL“)`
 - Muss kompatibel zu benutzten importierten Symbolen sein
- Sicherheit
 - Modul muss mit Quellen des zu benutzenden Kerns gebaut werden
 - Magic Numbers

kAPI & kABI

- Das Kernel API
 - kernel (application) programming interface
 - `EXPORT_SYMBOL ()`
 - `EXPORT_SYMBOL_GPL ()`
- Das Kernel ABI
 - kernel (application) binary interface

Kern Bau

Source installation

- Ftp kernel archive von www.kernel.org
- `$tar xvjf linux.v.x.y.z.tar.bz2 # „j“ = bunzip2`

Kern Bau

- GNU C-Compiler, Gcc (Cross-Compilation: Makefile anpassen)

```
$make config # or menuconfig, xconfig or gconfig  
$           # or defconfig  
$           # or edit .config & make oldconfig  
$make
```

Kernänderungen

- 1 Hole neuesten Kern
- 2 Folge LINUX Coding Style
- 3 Veröffentliche jeden Patch einzeln
- 4 Erkläre was er macht.
- 5 Beschreibe Dein Vertrauen in den Patch

Baue patch und wende ihn an

- Originalquellen in `linux-2.6.13.4`
- Modifizierte Quellen in `linux`
- `$ make -C linux-2.6.13.4 distclean`
- `$ make -C linux distclean`
- `$ diff -urN linux-2.6.13.4 linux > mypatch.diff`
- Wende Patch an (von root meines Linux Dirs)
- `$ cd linux-2-6-13-4`
- `$ patch -p1 < ../mypatch.diff`

Linux Codierungsstil

- Konventionen siehe `linux/Documentation/CodingStyle`
 - einrücken mit TAB (8 Zeichen)
 - Zeilenlänge: 80 Zeichen (aufgehoben?)
 - geschweifte Klammern a la K&R
 - Bezeichner: kurze treffende Namen
 - kurze übersichtliche Funktionen (<10 Parameter, <2Seiten)
 - Kommentare: *was nicht wie*
- Sonstiges
 - resource aquisition idiom
 - `gotos` not considered harmful (Fehlerbehandlung)
 - weitgehender Verzicht auf `typedef`, `#if` und `#ifdef`
 - macros, inline disease

Rule #1 in kernel programming: don't **ever** think that things actually work the way they are documented to work.

-- Linus Torvalds

Besonderheiten der Kernprogrammierung

- GNU C (gcc)
 - Inline Assembler
 - Inline Funktionen
 - Verzweigungsannotation
 - `if (unlikely(foo)) {...}`
- Keine Gleitkommazahlen
- keine Systembibliotheken (glibc)
 - String Funktionen in Kern:
 - `<linux/string.h>`
 - kein `printf`, aber `printk`
- Sehr kleiner Stack
 - keine Rekursion
 - kleine Parameterlisten
 - keine großen lokale Variable
- Keine glibc
- Kein Speicherschutz
- Parallelität
 - Nebenläufigkeit
 - SMP
 - Unterbrechungen
 - Kern Preemptiv
- Portabilität

Fehlerindikation

- in user space Funktionen (POSIX)
 - Rückgabewert: -1 entspricht Fehlerindikation
 - Fehlerart: errno
- In Kern Funktionen
 - Rückgabewert
 - ≥ 0 kein Fehler
 - < 0 (und > -4096): Fehler
 - siehe `<linux/errno.h>` wegen der Fehlerbedeutung

Beispiel printk()

- `kernel/printk.c`
- Gebrauch
 - Notfallmeldungen (z.B. panic)
 - debugging
 - allg. Informationen (z.B. Boot-Meldungen)
- Loglevel `<0> ... <7>` (cf. `include/linux/kernel.h`)
 - `DEFAULT_MESSAGE_LOGLEVEL`
 - aktueller Loglevel: `console_loglevel`
 - `DEFALUT_CONSOLE_LOGLEVEL`
 - kann mit `klogd -c` geändert werden
- Format: optionaler Loglevel + `printf`-Format
- merkt sich letzte `LOG_BUF_LEN` Zeichen in `log_buf`
 - wichtig falls Konsole noch nicht initialisiert