Inhalt

Das Applikations-Interface kurz vorgestellt...

Applikationsinterface

Applikationsschnittstelle für den Gerätezugriff

- Interfaces für Geräte- und Dateizugriff sind identisch.
- Pro Gerät mindestens eine Datei im Dateisystem (Gerätedatei)
- Systemfunktionen:

- open
- close
- read
- write
- ioctl
- select/poll
- fcntl
- seek

Open/Close

- „Öffnen“ des Gerätes:
  - Ist die Datei vorhanden/ist der Treiber überhaupt geladen?
  - Stimmen die Zugriffsrechte?
  - Welcher Zugriff auf das Gerät wird verlangt?
  - Rückgabewert: Filedeskriptor
- „Aufräumen“:
  - Freigeben der bei `open` allozierten Ressourcen.
  - Nach dem `close`-Aufruf befindet sich das Gerät wieder in einem definierten Zustand.

Read

```
ssize_t read( int fd, char *buffer, size_t max_bytes_to_read);
```

- Lesen von Daten (aus einer Datei oder aus der Hardware in den Speicherbereich der Applikation).
- Returnwert:
  - Anzahl der gelesenen Bytes (Minimum 1 Byte).
  - Fehlercode (-EINTR, -EAGAIN,...).

Write

- Schreiben von Daten (aus der Applikation in eine Datei oder auf die Hardware).
- Prototyp:
  - `ssize_t write( int fd, char *buffer, size_t MaxBytesToWrite);`
- Write: returniert die Anzahl der geschriebenen Bytes bzw. Fehlercode:
  - Soll eine definierte Anzahl Bytes geschrieben werden, muss in einer Schleife geschrieben werden.
  - Die Fehlerbedingung „Unterbrechung durch ein Signal“ (-`EINTR`) ist abzufangen (Aufruf wiederholen).

Ioctl

- Universal-Funktion zur Kommunikation mit dem Treiber.
- Unterstützt frei definierbare Kommandos mit frei definierbaren Parametern (Beispiel: ioctl zum „atomic“ write-read).
- Kommandos und Parameter werden durch den Treiber festgelegt.
- Prototype:
  - `int ioctl( int fd, int command, ...);`

Mmap

- Funktion, um Speicherbereiche eines Geräte (Hardware) in den Adressraum einer Applikation einzublenden.
- Damit kann die Applikation performant (ohne Übergang in den Kernel-Mode) auf Hardware (-Register) zugreifen. (Beispiel: xorg).
- Prototype:
  - void * mmap(void *start, size_t length, int prot, int flags, int fd, off_t offset);
  - int munmap(void *start, size_t length);

Select/poll

- Funktion um festzustellen, ob an einem oder an mehreren Geräten (Dateien oder Netzverbindungen) Daten zum Lesen bereit liegen bzw. Daten geschrieben werden können.
- Der Aufruf erfolgt zeitüberwacht.
- Mit dieser Systemfunktion kann eine Applikation ohne zu pollen mehrere Kanäle (Dateien) auf Ein- oder Ausgaben überwachen.

Select/poll

- Prototyp:
  - int select(int n, fd_set *read_fds, fd_set *write_fds, fd_set *exception_fds, struct timeval *timeout);
- Über die Makros FD_SET, FD_ZERO und FD_ISSET können die Datenstrukturen „read_fds“, „write_fds“ und „exception_fds“ initialisiert und später überprüft werden.

select/poll

```
#include <stdio.h>
#include <sys/time.h>
#include <sys/types.h>
#include <unistd.h>

int main(void)
{
    fd_set rfds;
    struct timeval tv;
    int retval;

    FD_ZERO(&rfds);
    FD_SET(0, &rfds);
    /* Wait up to five seconds. */
    tv.tv_sec = 5;
    tv.tv_usec = 0;

    retval = select(1, &rfds, NULL, NULL, &tv);
    /* Don't rely on the value of tv now! */

    if (retval)
        printf("Data is available now.\n");
        /* FD_ISSET(0, &rfds) will be true. */
    else
        printf("No data within five seconds.\n");

    exit(0);
}
```

lseek

- Mit dieser Funktion kann in einer Datei (bzw. bei einem Blockdevice) gezielt ein bestimmtes Byte gelesen bzw. geschrieben werden.
- Die Positionierung kann relativ
  - zum Dateianfang
  - zum Dateiende
  - zur aktuellen Position erfolgen.

Beispielapplikation: Auslesen des Bootsektors

```
int main( int argc, char **argv )
{
    int i, j, fd;
    unsigned char buffer[512];

    if( (fd=open(„/dev/hda“, O_RDONLY) )<0 ) {
        perror( „/dev/hda“ );
        exit( -1 );
    }
    if( read( fd, buffer, sizeof(buffer)) != sizeof(buffer) ) {
        perror( „read“ );
        exit( -2 );
    }
    for( i=0; i<sizeof(buffer)/16); i++ ) {
        for( j=0; j<16; j++ ) {
            printf( „%2.2x „,buffer[i*16+j] );
        }
        printf(„\n“);
    }
    close( fd );
    return 0;
}
```

Beispielapplikation: Ausgabe auf den Drucker

```
int main( int argc, char **argv )
{
   int fd;
   char *ptr="aaaaaa";

   if( (fd=open(„/dev/lp0", O_WRONLY))<0 ) {
      perror(„/dev/lp0");
      exit(-1);
   }
   if( write(fd,ptr,strlen(ptr))<strlen(ptr)) {
      perror(„write");
   }
   close( fd );
   return 0;
}
```

Kontrollfluss

- Für den Zugriff auf ein Gerät gibt es zwei Zugriffsarten:
  - blocking mode
  - non blocking mode
- Beim „blocking mode“ wird eine Applikation in den Zustand „schlafend“ versetzt, falls beim read-Aufruf angeforderte Daten noch nicht zur Verfügung stehen.
- Beim „blocking mode“ wird eine Applikation in den Zustand „schlafend“ versetzt, falls bei einem write-Aufruf auszugebende Daten noch nicht geschrieben werden können.

Kontrollfluss

```
#include <stdio.h>

int main( int argc, char **argv )
{
    int fd=0; // stdin, standardmaessig im blocking mode
    char buffer[512];

    printf("Blocking-Mode: WARTEN AUF EINGABE ...\n");

    read( fd, buffer, sizeof(buffer) );  // blockierendes Warten

    printf("Eingabe getaetigt...\n");
    return 0;
}
```

Kontrollfluss

- Beim „non-blocking mode“ kommt der Aufruf (Systemcall) direkt zurück, auch wenn keine Daten gelesen bzw. geschrieben werden konnten.
- Dieser Fall ist am Fehlercode -EAGAIN erkenntlich.
- Der Zugriffsmode wird beim Zugriff auf Dateien entweder
  - beim Öffnen angegeben oder
  - nach dem Öffnen mit der Systemfunktion „fcntl“.

Kontrollfluss

```
#include <stdio.h>
#include <fcntl.h>

int main( int argc, char **argv )
{
    int fd=0; // stdin, standardmaessig im blocking mode
    int ret;
    char buffer[512];

    fcntl( fd, F_SETFL, O_NONBLOCK);
    printf("Nonblocking-Mode: KEIN WARTEN\n");

    ret=read( fd, buffer, sizeof(buffer) );

    printf("Returnwert: %d\n", ret);
    return 0;
}
```

Zusammenfassung

- Das einheitliche Interface für den Zugriff auf Dateien oder auf Geräte entlastet den Programmierer.
- Der Datenfluss ist relativ simpel programmiert.
- Der Kontrollfluss (blocking/non blocking) ist wesentlich komplexer.
- Iocontrols bieten dem Treiberentwickler sehr viel Freiheit, sind aber fehlerträchtig und sollten vermieden werden.

Den Kernel erweitern ...

Infos:

- Quade/Kunst: Linux-Treiber entwickeln. 3. Auflage, dpunkt-Verlag, 2006, S. 67-86.

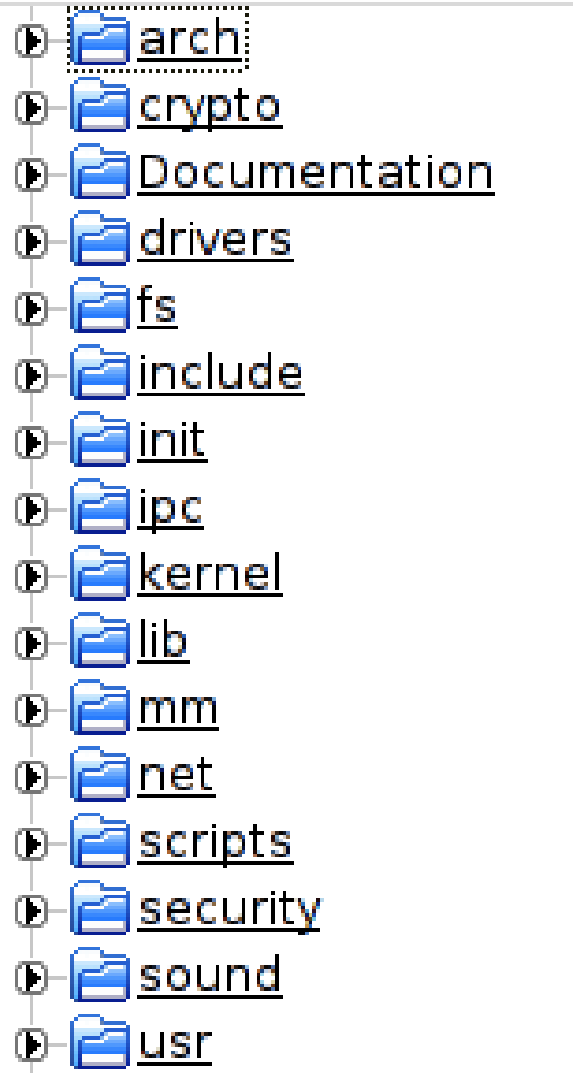
Bevor es losgeht ...

- Voraussetzungen für die Kernelprogrammierung:
  - Kernelquellen
    - unter /usr/src/linux
  - Kernel-Konfiguration
  - Programme „insmod“, „rmmod“

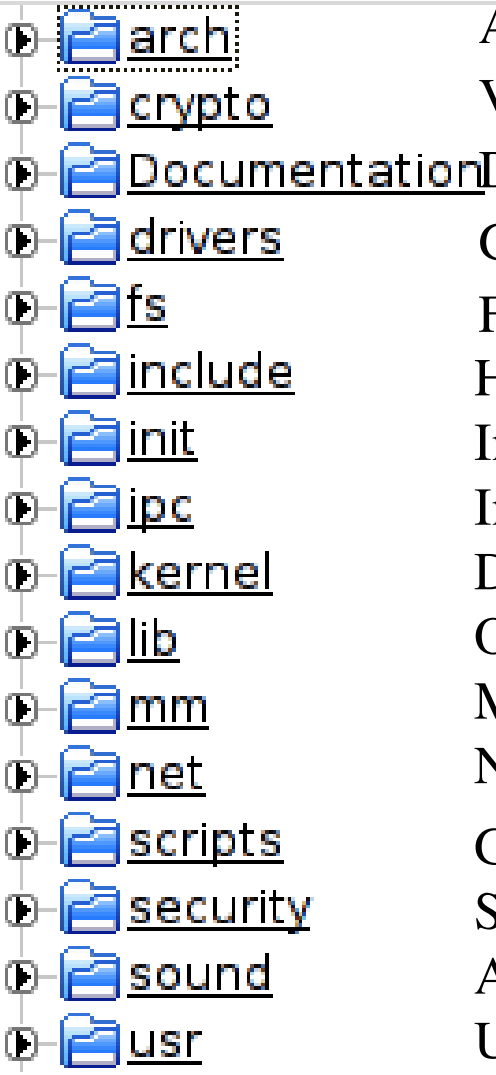
Voraussetzungen

Aufbau der Kernel-Quellen

- Download über http://www.kernel.org
- Ablage unter /usr/src/linux-<version>
  - `/usr/src/linux-2.6.30`
- Link der jeweils aktuellen Version auf /usr/src/linux



Aufbau der Kernel-Quellen



arch	Architekturspezifischer Code
crypto	Verschlüsselungsalgorithmen (IPSec)
Documentation	Dokumentation
drivers	Gerätetreiber
fs	Filesysteme
include	Header-Dateien
init	Initialisierung des Kernels
ipc	Interprozesskommunikation
kernel	Der Kern des Kerns
lib	Oft benötigte Hilfsfunktionen
mm	Memory-Management
net	Netzwerk Code
scripts	Generierungs-Skripte
security	Sicherheits-Module (LSM)
sound	Audio-Unterstützung
usr	User-Software des Init-Filesystems

Kernel-Erweiterungen

- Können als Systemcalls realisiert werden.
  - Negativ: Kompatibilitätsprobleme
- Können als Treiber realisiert werden.
  - Built-In Code
  - ladbare Module

Für den Programmierer ist diese Unterscheidung zunächst unerheblich.

Kurze Zusammenfassung

- Der Kernel wird über Treiber erweitert.
- Treiber werden als zur Laufzeit ladbare Module oder als fester Bestandteil des Kernels realisiert.
- Es lassen sich virtuelle und reale Geräte implementieren.
- Der Zugriff auf Geräte wird meist in Form eines zeichenorientierten Gerätes realisiert (Char-Dev).

Gerätetreiber-Übersicht

Kernel-Module

- Treiber sind (meist) Module
- Module werden dynamisch geladen:
  - Geringerer Ressourcen-Verbrauch im Kernel.
  - Einfache Treiberentwicklung (es ist kein ständiges Rebooten notwendig).
- „`insmod`“ lädt das Modul (den Treiber)
- „`rmmod`“ entlädt das Modul (den Treiber)
- „`lsmod`“ listet die geladenen Module

Modul-Code

```
#include <linux/version.h>
#include <linux/module.h>

MODULE_LICENSE("GPL");

int init_module(void)
{
   printk("init_module called\n");
   return 0;
}

void cleanup_module(void)
{
   printk("cleanup_module called\n");
}
```

Aufgabe der Funktion `init_module`:

- Treiber beim IO-Subsystem anmelden
- Treiber bei sonstigen Subsystemen anmelden
  - PCI
  - Sys
  - ...
- Initialisierung von Treiberobjekten (z.B. Waitqueues)
- Eventuell Geräteinitialisierung (Reservieren von Ressourcen)

Aufgabe der Funktion `cleanup_module`:

- Freigeben der allozierten Systemressourcen
- Abmelden des Moduls beim System

Lizenzangabe

- Module sollen angeben, welcher Lizenz sie unterliegen.
- Dazu dient das Makro „MODULE_LICENSE().
- Folgende Lizenzen stehen zur Auswahl:
  - „GPL“
  - „GPL and additional rights“
  - „Dual BSD/GPL“
  - „Dual MPL/GPL“
  - „Proprietary“

Lizenzen

- Besitzt ein Modul zwei Lizenzen, zählt die GPL.
- Das Kommando „modinfo“ zeigt die Lizenz an.
  - `modinfo -l modulename.ko`
- Fehler, die aus einem proprietären Modul herrühren, werden nicht verfolgt.
- Hersteller können sich auf Fehler ihrer Module beschränken.

Kurzer Exkurs zu Debugausgaben

- Im Kernel gibt es die zu `printf()` korrespondierende Funktion `printk()`.
- Ausgaben von `printk()` werden im Syslog protokolliert.
- Die verschiedenen Syslog-Level (`LOG_ALERT, LOG_INFO, LOG_NOTICE, LOG_DEBUG`,...) werden über die drei ersten Zeichen des Formatstrings gesteuert.

Debugausgaben

Symbol	Wert	Bedeutung
KERN_EMERG	<0>	Das System ist nicht mehr zu gebrauchen.
KERN_ALERT	<1>	Sofortige Maßnahmen sind erforderlich.
KERN_CRIT	<2>	Der Systemzustand ist kritisch.
KERN_ERR	<3>	Fehlerzustände sind aufgetreten.
KERN_WARNING	<4>	Warnung.
KERN_NOTICE	<5>	Wichtige Nachricht, kein Fehler.
KERN_INFO	<6>	Zur Information.
KERN_DEBUG	<7>	Debug-Information.

Beispiel:

```
printk("<7>user will %d bytes kopieren.\n", count );
printk(KERN_DEBUG "user will %d bytes kopieren.\n",count);
```

Debugausgaben

- Über `pr_debug()` und `pr_info()` sind übersichtliche Ausgaben (bei bedingter Compilierung) möglich:

```
    pr_debug( "Lesefifo=%d\n", fifoindex );

    // entspricht:

#ifdef DEBUG
    printk( KERN_DEBUG "Lesefifo=%d\n", fifoindex );
#endif
```

```
    pr_info( "Treiber erfolgreich geladen.\n");
    // entspricht:
    printk( KERN_INFO "Treiber erfolgreich geladen.\n");
```

Debugausgaben

- Zusätzlich gibt es noch Funktionen, die einen Stacktrace ausgeben:
  - `WARN();` -> Code wird weiterbearbeitet.
  - `WARN_on( bedingung );` -> Code wird weiterbearbeitet.
  - `WARN_on_once(bedingung);`-> Code wird weiterbearbeitet.
  - `BUG();` -> Code wird abgebrochen.
  - `BUG_ON( bedingung );` -> Code wird abgebrochen.

ENDE DES EXKURSES.

Makefile

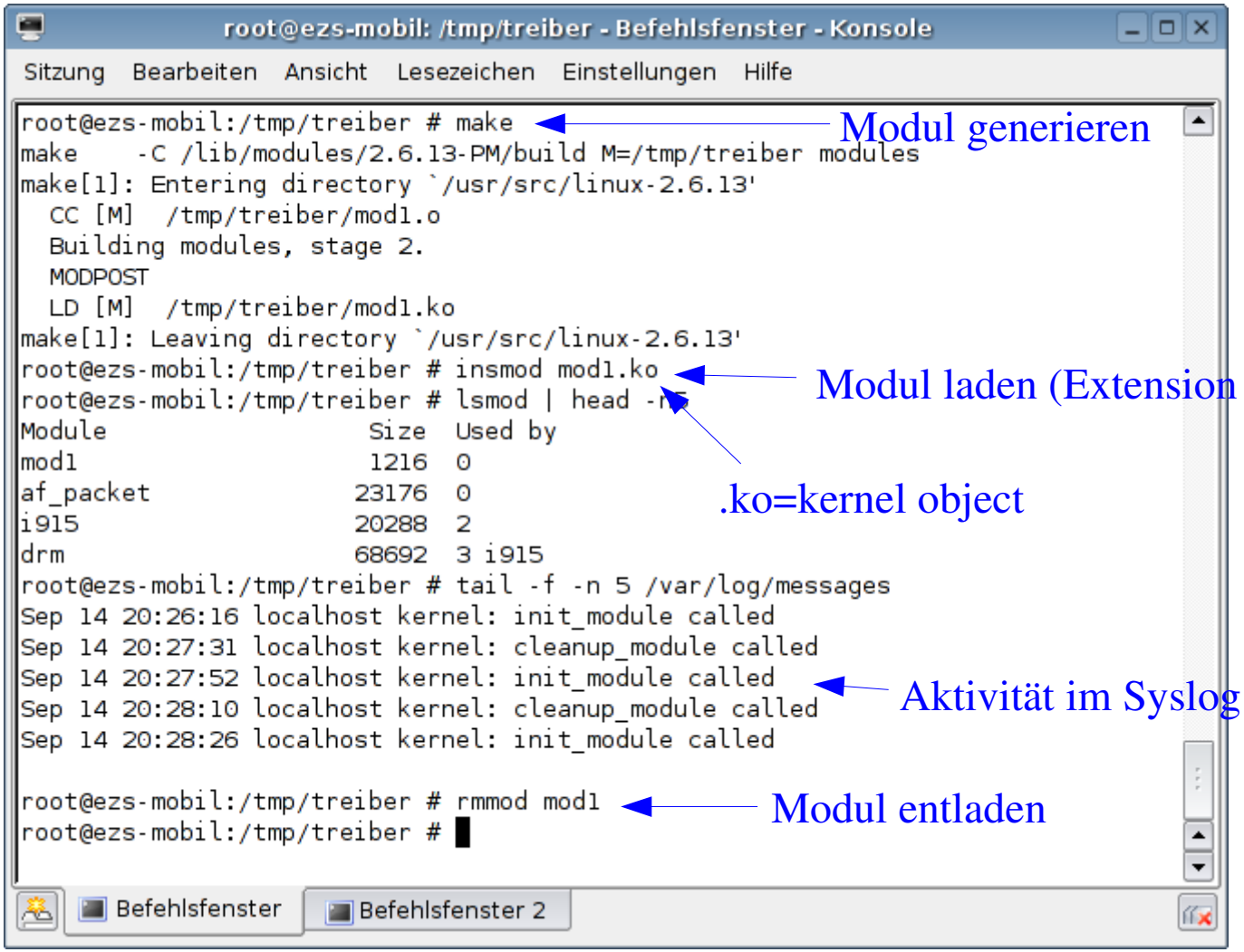
```
ifneq ($(KERNELRELEASE),)
obj-m := mod1.o

else
KDIR  := /lib/modules/$(shell uname -r)/build
PWD   := $(shell pwd)

default:
   $(MAKE)  -C $(KDIR)  M=$(PWD) modules
endif
```

Erste Schritte

Modul-Test



Übung 1.2: Kernel-Modul

- Tippen Sie das Codegerüst eines einfachen Moduls (Funktionen `init_module()`, `cleanup_module()`) ein (Name `mod1.c`).
- Geben Sie das zugehörige Makefile ein.
- Compilieren Sie das Modul.
- Geben Sie in einer (zweiten) Konsole das Kommando `tail -f /var/log/messages` ein.
- Laden und entladen Sie das Modul:
  - `insmod mod1.ko`
  - `lsmod`
  - `rmmod mod1`

Linux Kernelmodul-Template

```
#include <linux/module.h>
#include <linux/init.h>

static int __init mod_init(void)
{
    printk("mod_init called\n");
    return 0;
}

static void __exit mod_exit(void)
{
    printk("mod_exit called\n");
}

module_init( mod_init );
module_exit( mod_exit );

// Metainformation
MODULE_AUTHOR("Juergen Quade");
MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("Just a Modul-Template, without specific functionality.");
MODULE_SUPPORTED_DEVICE("none");
```

Diese Makros abstrahieren den Unterschied zwischen Modul- und Built-In-Code.

Übung 1.3: Template

- Modifizieren Sie das Codegerüst, so dass es als Template für die folgenden Übungen eingesetzt werden kann.
- Generieren und testen Sie das Template.