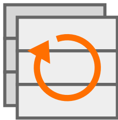


Operating
System
Group

TUHH
Technische Universität Hamburg

Betriebssystembau (BSB)

VL 11 – Gerätetreiber

Christian Dietrich

Operating System Group

SS 22 – 5. Juli 2022

Interaktion während der Veranstaltung

TUHH
Technische Universität Hamburg

- BSB ist vom "Stil" her eine **interaktive Präsenzveranstaltung**
  - Wir wollen versuchen, dieses soweit wie möglich "online" zu retten

⇝ **Synchrones** Format – Fragen und Beteiligung ist erwünscht!

- Interaktion **während** der Veranstaltung
  - 1. „Melden“
  - 2. „Drankommen“
  - 3. Profit
- Interaktion **außerhalb** der Veranstaltung
  - Über das Stud.IP-Forum
  - **NEU**: EIM Mattermost Team: `https://communicating.tuhh.de/eim`

Aufzeichnung der Vorlesung

You are on mic!

- Auf vielfachen Studierendenwunsch:
  Veranstaltung wird aufgezeichnet
  - Wird im Anschluss über Stud.IP verfügbar gemacht
  - ⤳ Geschlossene Nutzergruppe
- Aufgezeichnet wird
  - Screencast der BBB-Session **ohne den Chat (Klarnamen)**
  - Ihre Stimme bei Fragen und Anmerkungen
  - **Durch Aktivierung Ihres Mikrofons willigen Sie dazu ein!**
- Fragen können über direkte Nachricht an mich auch anonym gestellt werden

Überblick: Einordnung der VL

TUHH Technische Universität Hamburg

Agenda

TUHH
Technische Universität Hamburg

- Einordnung
  - Bedeutung von Gerätetreibern
- Anforderungen an das BS
  - Namensraum
  - Einheitlicher Zugriff
  - Spezifischer Zugriff
- Struktur des E/A-Systems
  - Treibermodell
  - Linux
  - Windows
- Zusammenfassung

Agenda

TUHH
Technische Universität Hamburg

Einordnung
- Bedeutung von Gerätetreibern

Anforderungen an das BS
- Namensraum
- Einheitlicher Zugriff
- Spezifischer Zugriff

Struktur des E/A-Systems
- Treibermodell
- Linux
- Windows

Zusammenfassung

Bedeutung von Gerätetreibern (1)

TUHH
Technische Universität Hamburg

- Anteil an Treibercode in Linux 5.2.4

```
costa :: ~/Downloads/linux-master % du -k -d 1 . | sort -n
18532  ...
4896    ./lib
9524    ./kernel
30620   ./net
31108   ./tools
37504   ./sound
40336   ./fs
42596   ./include
45576   ./Documentation
129660  ./arch
558928  ./drivers
```

(torvalds-tree, 25.6.2019)

Bedeutung von Gerätetreibern (2)

TUHH
Technische Universität Hamburg

Bedeutung von Gerätetreibern (3)

TUHH
Technische Universität Hamburg

- In Linux (v5.2.4) ist der Treibercode (ohne ./arch) etwa **60 mal so groß** wie der Code des Kernels
  - Und wächst rasant!
    - Bei v.3.2.1 waren es noch: 50 mal mehr
    - Bei v2.6.32 waren es noch: 25 mal mehr
    - Bei v.2.6.11 waren es noch: 10 mal mehr
  - Windows unterstützt ebensoviele Geräte...
- Treiberunterstützung ist **entscheidender Faktor** für die Akzeptanz eines Betriebssystems
  - In Gerätetreibern steckt eine erhebliche Arbeitsleistung

Bedeutung von Gerätetreibern (3)

- In Linux (v5.2.4) ist der Treibercode (ohne ./arch) etwa **60 mal so groß** wie der Code des Kernels
  - Und wächst rasant!
    - Bei v.3.2.1 waren es noch: 50 mal mehr
    - Bei v2.6.32 waren es noch: 25 mal mehr
    - Bei v.2.6.11 waren es noch: 10 mal mehr
  - Windows unterstützt ebensoviele Geräte...
- Treiberunterstützung ist **entscheidender Faktor** für die Akzeptanz eines Betriebssystems
  - In Gerätetreibern steckt eine erhebliche Arbeitsleistung

Der Entwurf des E/A-Subsystems erfordert viel Geschick!

- Treiberinfrastruktur mit vielen wiederverwendbare Funktionen
- Treibermodell mit klaren Vorgaben zu Struktur und Verhalten

Letztlich geht es darum, die Entwicklung guter Treiber so einfach wie möglich zu machen.

Agenda

TUHH
Technische Universität Hamburg

Einordnung
- Bedeutung von Gerätetreibern

Anforderungen an das BS
- Namensraum
- Einheitlicher Zugriff
- Spezifischer Zugriff

Struktur des E/A-Systems
- Treibermodell
- Linux
- Windows

Zusammenfassung

Anforderungen an Betriebssysteme

TUHH
Technische Universität Hamburg

- Ressourcenschonender Umgang mit Geräten
  - schnell arbeiten
  - Energie sparen
  - Speicher, *Ports* und *Interrupt*-Vektoren sparen
  - Aktivierung und Deaktivierung zur Laufzeit
  - Generische *Power Management* Schnittstelle
- Einheitlicher Zugriffsmechanismus
  - **minimaler Satz von Operationen** für verschiedene Gerätetypen
  - **mächtige Operationen** für vielfältige Typen von Anwendungen
- auch gerätespezifische Zugriffsfunktionen

Linux – einheitlicher Zugriff (1)


```
echo "Hallo, Welt" > /dev/ttyS0
```

- Geräte sind über Namen im Dateisystem ansprechbar
- Vorteile:
  - Systemaufrufe für Dateizugriff (`open`, `read`, `write`, `close`) können auch für sonstige E/A verwendet werden
  - Zugriffsrechte können über die Mechanismen des Dateisystems gesteuert werden
  - Anwendungen sehen keinen Unterschied zwischen Dateien und "Gerätedateien"
- Probleme:
  - blockorientierte Geräte müssen in Byte-Strom verwandelt werden
  - manche Geräte lassen sich nur schwer in dieses Schema pressen
    - Beispiel: 3D Graphikkarte

Linux – einheitlicher Zugriff (2)

- blockierende Ein-/Ausgabe (Normalfall)
  - `read`: Prozess blockiert bis die angeforderten Daten da sind
  - `write`: Prozess blockiert bis Schreiben möglich ist
- nicht-blockierende Ein-/Ausgabe
  - `open`/`read`/`write` mit dem Zusatz-Flag `O_NONBLOCK`
  - statt zublockieren kehren `read` und `write` so mit `-EAGAIN` zurück
  - der Aufrufer kann/muss die Operation später wiederholen
- nebenläufige Ein-/Ausgabe
  - `aio_(read|write|...)` (POSIX1003.1-2003), `io_uring` (Linux)
  - indirekt mittels Kindprozess (fork/join)
  - `select`, `poll` Systemaufrufe

Windows – einheitlicher Zugriff (1)

TUHH
Technische Universität Hamburg

- Geräte sind Kern-Objekte der Executive

Windows – einheitlicher Zugriff (2)

TUHH
Technische Universität Hamburg

WinObj - Sysinternals: www.sysinternals.com

File Edit Find View Options Help

Quick Find: Search

- \
  - ArcName
  - BaseNamedObjects
  - Callback
  - Device
  - Driver
  - FileSystem
  - GLOBAL??
  - KernelObjects
  - KnownDlls
  - KnownDlls32
  - NLS
  - ObjectTypes
  - RPC Control
  - Security
  - Sessions
  - Windows

Name	Type	Symbolic Link Target
A:	SymbolicLink	\Device\Floppy0
ACPI#FixedButton#2&daba3ff&1#{4afa3d53-74a7-11d0-be5...	SymbolicLink	\Device\00000012
ACPI#PNP0303#4&3b4f7da4&0#{884b96c3-56ef-11d1-bc8c...	SymbolicLink	\Device\00000018
ACPI#PNP0400#4&3b4f7da4&0#{97f76ef0-f883-11d0-af1f-0...	SymbolicLink	\Device\0000001b
ACPI#PNP0501#1#{4d36e978-e325-11ce-bfc1-08002be10318}	SymbolicLink	\Device\0000001c
ACPI#PNP0501#1#{86e0d1e0-8089-11d0-9ce4-08003e301f73}	SymbolicLink	\Device\0000001c
ACPI#PNP0F13#4&3b4f7da4&0#{378de44c-56ef-11d1-bc8c...	SymbolicLink	\Device\00000019
ACPI_ROOT_OBJECT	SymbolicLink	\Device\0000000e
ahcache	SymbolicLink	\Device\ahcache
AUX	SymbolicLink	\DosDevices\COM1
BitLocker	SymbolicLink	\Device\BitLocker
C:	SymbolicLink	\Device\HarddiskVolume2
CdRom0	SymbolicLink	\Device\CdRom0
COM1	SymbolicLink	\Device\Serial0
CON	SymbolicLink	\Device\ConDrv\Console
CONIN$	SymbolicLink	\Device\ConDrv\CurrentIn
CONOUT$	SymbolicLink	\Device\ConDrv\CurrentOut
Disk{f0199285-8ef8-659c-3937-6ad71202b753}	SymbolicLink	\Device\Harddisk0\DR0
DISPLAY#Default_Monitor#4&77741e3&0&UID0#{866519b5...	SymbolicLink	\Device\00000020
DISPLAY#Default_Monitor#4&77741e3&0&UID0#{e6f07b5f-...	SymbolicLink	\Device\00000020
DISPLAY1	SymbolicLink	\Device\Video0
DISPLAY2	SymbolicLink	\Device\Video1
F:	SymbolicLink	\Device\CdRom0
FDC#GENERIC_FLOPPY_DRIVE#5&e9e2334&0&0#{53f5630d...	SymbolicLink	\Device\FloppyPDO0
FDC#GENERIC_FLOPPY_DRIVE#5&e9e2334&0&0#{53f56311...	SymbolicLink	\Device\FloppyPDO0

Windows – einheitlicher Zugriff (2)

TUHH
Technische Universität Hamburg

WinObj - Sysinternals: www.sysinternals.com

File Edit Find View Options Help

Quick Find: Search

- \
  - ArcName
  - BaseNamedObjects
    - Restricted
  - Callback
  - **Device**
    - Http
    - Ide
    - Harddisk0
  - Driver
  - FileSystem
  - GLOBAL??
  - KernelObjects
  - KnownDlls
  - KnownDlls32
  - NLS
  - ObjectTypes
  - RPC Control
  - Security
  - Sessions
    - 0
    - 1
    - 2
    - BNOLINKS
  - Windows
    - WindowStations

Name	Type	Symbolic Link Target
FakeVid14	Device	
FakeVid15	Device	
FakeVid2	Device	
FakeVid3	Device	
FakeVid4	Device	
FakeVid5	Device	
FakeVid6	Device	
FakeVid7	Device	
FakeVid8	Device	
FakeVid9	Device	
FileInfo	Device	
Floppy0	Device	
FloppyPDO0	Device	
FsWrap	Device	
gpuenergydrv	Device	
HarddiskVolume1	Device	
HarddiskVolume2	**Device**	
Ip	SymbolicLink	\Device\Tdx
Ip6	SymbolicLink	\Device\Tdx
IPSECDOSP	Device	
KeyboardClass0	Device	
KeyboardClass1	Device	
KeyboardClass3	Device	
KeyboardClass5	Device	
KeyboardClass7	Device	
KeyboardClass9	Device	

Windows – einheitlicher Zugriff (3)

TUHH Technische Universität Hamburg

- synchrone oder asynchrone Ein-/Ausgabe

```
BOOL ReadFile(
  HANDLE hFile,
  LPVOID lpBuffer,
  DWORD nNumberOfBytesToRead,
  LPDWORD lpNumberOfBytesRead,
  LPOVERLAPPED lpOverlapped
);
```

NULL: synchrones Lesen

```
BOOL GetOverlappedResult(
  HANDLE hFile,
  LPOVERLAPPED lpOverlapped,
  LPDWORD lpNumberOfBytesTransferred,
  BOOL bWait
);
```

true: auf Ende warten
false: Status erfragen

- weitere Möglichkeiten:
  - E/A mit *Timeout*
  - WaitForMultipleObjects – warten auf 1–N Kernobjekte
    - Datei-Handles, Semaphore, Mutex, Thread-Handle, ...
  - I/O Completion Ports
    - Aktivierung eines wartenden Threads nach I/O Operation

Linux – gerätespez. Funktionen (1)

TUHH
Technische Universität Hamburg

- spezielle Geräteeigenschaften werden (klassisch) über **ioctl** angesprochen:

```
IOCTL(2)            Linux Programmer's Manual            IOCTL(2)

NAME
       ioctl - control device

SYNOPSIS
       #include <sys/ioctl.h>

       int ioctl(int d, int request, ...);
```

- Schnittstelle generisch und Semantik gerätespezifisch:

```
CONFORMING TO
       No single standard. Arguments, returns, and semantics of
       ioctl(2) vary according to the device driver in question
       (the call is used as a catch-all for operations that
       don't cleanly fit the Unix stream I/O model). The ioctl
       function call appeared in Version 7 AT&T Unix.
```

Linux – gerätespez. Funktionen (2)

Linux 2.6 – das **Gerätemodell im sys-Dateisystem**

Geräte sind virtuelle Verzeichnisse

Symbolische *Links* ermöglichen unterschiedliche Sortierkriterien

virtuelle Dateien repräsentieren die Geräte- bzw. Treiber-**Attribute**.

/sys/

block
bus
cdev
class
devices
firmware
power

hd0
hda
hdc
ram0

ide
pci
pcmcia
platform
pnp
usb

input
net
pcmcia_socket
pty
usb
usb_host

legacy
pci0000:00
pnp0
pnp1
system

acpi

disk state

hda1
dev
device
queue
range
size
stat

devices
drivers

devices
drivers

eth0
lo

0000:00:00.0
0000:00:01.0
0000:00:04.0
0000:00:04.1
0000:00:04.2
0000:00:04.3
0000:00:06.0
0000:00:06.1
0000:00:08.0
0000:01:00.0
0000:01.00.1
0000:02:00.0

3c59x
Neo Magic 256
PCI IDE
PIIX IDE
agpgart_ali
agpgart_amdk7
ehci_hcd
ohcihcd
parport_pc
serial
uhcihcd
yenta_cardbus

addr_len
address
broadcast
device
driver
features
flags
ifindex
iflink
mut
statistics
tx_queue_len

class
config
device
irq
name
pools
power
resource
subsystem_device
subsystem_vendor
usb1
vendor

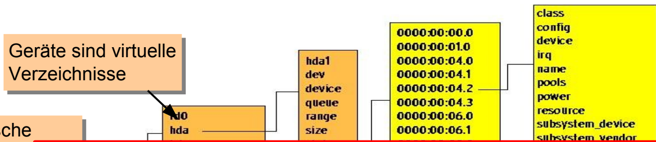
Blockgeräte und deren Attribute
Bussysteme mit den daran angeschlossenen Geräten
Zeichenorientierte Geräte und deren Attribute
Geräte nach Geräteklassen sortiert
Sonstige Geräte
Schnittstelle für Firmware-Downloads

Quelle:

Linux – gerätespez. Funktionen (2)

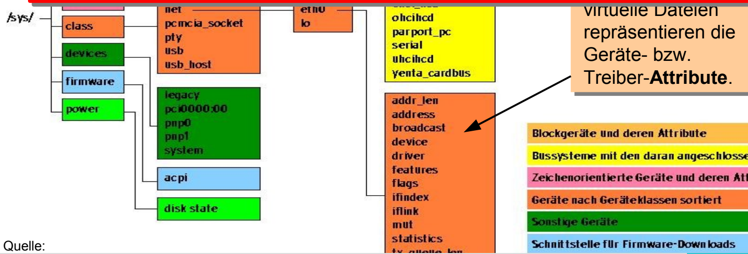
TUHH Technische Universität Hamburg

Linux 2.6 – das **Gerätemodell im sys-Dateisystem**



Symbolische *Links* ermögl
unterschiedli
Sortierkriteri

Das Gerätemodell erlaubt Kern- und Anwendungsfunktionen, die Rechnerhardware zu erforschen. Beispielsweise kann eine Power-Management-Funktion abhängige Geräte in der richtigen Reihenfolge stoppen und starten.



Virtuelle Dateien repräsentieren die Geräte- bzw. Treiber-**Attribute**.

- Blockgeräte und deren Attribute
- Bussysteme mit den daran angeschlossenen Geräten
- Zeichenorientierte Geräte und deren Attribute
- Geräte nach Geräteklassen sortiert
- Sonstige Geräte
- Schnittstelle für Firmware-Downloads

Quelle:

Windows – gerätespez. Funktionen

TUHH
Technische Universität Hamburg

- **DeviceIoControl** entspricht dem UNIX ioctl:

```
BOOL DeviceIoControl(
  HANDLE hDevice,
  DWORD dwIoControlCode,
  LPVOID lpInBuffer,
  DWORD nInBufferSize,
  LPVOID lpOutBuffer,
  DWORD nOutBufferSize,
  LPDWORD lpBytesReturned,
  LPOVERLAPPED lpOverlapped
);
```

Kommunikation über untypisierte Puffer direkt mit dem Treiber

auch asynchron möglich

- und was sonst?
  - alle Geräte und Treiber werden durch Kern-Objekte repräsentiert
    - spezielle Systemaufrufe gestatten das Erforschen dieses Namensraums
  - statische Konfigurierung erfolgt über die Registry
  - dynamische Konfigurierung erfolgt z.B. über WMI
    - *Windows Management Instrumentation*

Agenda

TUHH
Technische Universität Hamburg

Einordnung
- Bedeutung von Gerätetreibern

Anforderungen an das BS
- Namensraum
- Einheitlicher Zugriff
- Spezifischer Zugriff

Struktur des E/A-Systems
- **Treibermodell**
- **Linux**
- **Windows**

Zusammenfassung

Struktur des E/A Systems (1)

TUHH
Technische Universität Hamburg

- Treiber mit unterschiedlicher Schnittstelle ...

  - erlauben die volle Ausnutzung aller Geräteeigenschaften
  - erfordern eine Erweiterung des E/A Systems für jeden Treiber
    - enormer Aufwand bei der heutigen Gerätevielfalt
    - unrealistisch, da erst das BS und dann die Treiber entstehen

Struktur des E/A Systems (2)

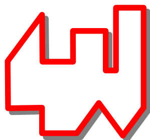
- Treiber mit uniformer Schnittstelle ...

geräteunabhängige E/A Funktionen
E/A Anforderungen
E/A Beendingung
Treiber
E/A System
Unterbrechung
Zugriff auf Hardware-Ressourcen
Hardwarezugriffsfunktionen
Festplatte
Drucker
Tastatur

  - ermöglichen ein (dynamisch) erweiterbares E/A System
  - erlauben flexibles "Stapeln" von Gerätetreibern
    - virtuelle Geräte
    - Filter

Das Treibermodell umfasst. . .

TUHH
Technische Universität Hamburg



"detaillierte Vorgaben für die Treiber-Entwicklung"

- die Liste der erwarteten Treiber-Funktionen
- Festlegung optionaler und obligatorischer Funktionen
- die Funktionen, die ein Treiber nutzen darf
- Interaktionsprotokolle
- Synchronisationsschema und Funktionen

- Festlegung von **Treiberklassen** falls mehrere Schnittstellentypen unvermeidbar sind

Anforderungen an Treiber

- Zuordnung zu Gerätedateien erlauben
- Verwaltung mehrerer Geräteinstanzen
- Operationen:
  - Hardware-Erkennung
  - Initialisierung und Beendigung
  - Lesen und Schreiben von Daten
    - ggf. auch *Scatter/Gather*
  - Steueroperationen und Gerätestatus
    - z.B. über ioctl oder virtuelles Dateisystem
  - Energieverwaltung
- intern zu bewältigen:
  - Synchronisation
  - Pufferung
  - Anforderung benötigter Systemressourcen

Linux – Treibergerüst: Registrierung

TUHH
Technische Universität Hamburg

```
MODULE_AUTHOR("B.S. Student");
MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("Dummy Treiber.");
MODULE_SUPPORTED_DEVICE("none");

static struct file_operations fops;
// ... Initialisierung von fops (Funktionszeiger)

static int __init mod_init(void){
  if(register_chrdev(240,"DummyDriver",&fops)==0)
    return 0; // Treiber erfolgreich angemeldet
  return -EIO; // Anmeldung beim Kernel fehlgeschlagen
}

static void __exit mod_exit(void){
  unregister_chrdev(240,"DummyDriver");
}

module_init( mod_init );
module_exit( mod_exit );
```

Metainformation, anzufragen mit 'modinfo'

Registrierung für das char-Device mit der **Major-Number 240**

mod_init und mod_exit werden beim Laden bzw. Entladen ausgeführt.

Linux – Treibergerüst: Operationen

TUHH Technische Universität Hamburg

```
static char hello_world[]="Hello World\n";

static int dummy_open(struct inode *geraete_datei,
  struct file *instanz) {
    printk("driver_open called\n"); return 0;
}

static int dummy_close(struct inode *geraete_datei,
  struct file *instanz) {
    printk("driver_close called\n"); return 0;
}

static ssize_t dummy_read(struct file *instanz,
  char *user, size_t count, loff_t *offset ) {
    int not_copied, to_copy;
    to_copy = strlen(hello_world)+1;
    if( to_copy > count ) to_copy = count;
    not_copied=copy_to_user(user,hello_world,to_copy);
    return to_copy-not_copied;
}

static struct file_operations fops = {
  .owner   =THIS_MODULE,
  .open    =dummy_open,
  .release=dummy_close,
  .read    =dummy_read,
};
```

die Treiberoperationen entsprechen den normalen Dateioperationen

in diesem Beispiel machen open und close nur Debugging-Ausgaben

mit **copy_to_user** und **copy_from_user** kann man Daten zwischen Kern- und Benutzer-adressraum austauschen

es gibt noch wesentlich mehr Operationen, sie sind jedoch größtenteils optional

Linux – Treibergerüst: Operationen


```
// Struktur zur Einbindung des Treibers in das virtuelle Dateisystem
struct file_operations {
    struct module *owner;
    loff_t (*llseek) (struct file *, loff_t, int);
    ssize_t (*read) (struct file *, char __user *, size_t, loff_t *);
    ssize_t (*aio_read) (struct kiocb *, char __user *, size_t, loff_t);
    ssize_t (*write) (struct file *, const char __user *, size_t, loff_t *);
    ssize_t (*aio_write) (struct kiocb *, const char __user *, size_t, loff_t);
    int (*readdir) (struct file *, void *, filldir_t);
    unsigned int (*poll) (struct file *, struct poll_table_struct *);
    int (*ioctl) (struct inode *, struct file *, unsigned int, unsigned long);
    int (*mmap) (struct file *, struct vm_area_struct *);
    int (*open) (struct inode *, struct file *);
    int (*flush) (struct file *);
    int (*release) (struct inode *, struct file *);
    int (*fsync) (struct file *, struct dentry *, int datasync);
    int (*aio_fsync) (struct kiocb *, int datasync);
    int (*fasync) (int, struct file *, int);
    int (*lock) (struct file *, int, struct file_lock *);
    ssize_t (*readv) (struct file *, const struct iovec *, unsigned long, loff_t *);
    ssize_t (*writev) (struct file *, const struct iovec *, unsigned long, loff_t *);
    ssize_t (*sendfile) (struct file *, loff_t *, size_t, read_actor_t, void __user *);
    ssize_t (*sendpage) (struct file *, struct page *, int, size_t, loff_t *, int);
    unsigned long (*get_unmapped_area)(struct file *, unsigned long,
        unsigned long, unsigned long, unsigned long);
};
```

Linux – Treiber-Infrastruktur

TUHH
Technische Universität Hamburg

- Ressourcen reservieren
  - Speicher, Ports, IRQ-Vektoren, DMA Kanäle
- Hardwarezugriff
  - Ports und Speicherblöcke lesen und schreiben
- Speicher dynamisch anfordern
- Blockieren und Wecken von Prozessen im Treiber
  - waitqueue
- Interrupt-Handler anbinden
  - low-level
  - Tasklets für länger dauernde Aktivitäten
- Spezielle APIs für verschiedene Treiberklassen
  - Zeichenorientierte Geräte, Blockgeräte, USB-Geräte, Netzwerktreiber
- Einbindung in das `proc` oder `sys` Dateisystem

Windows – E/A System

TUHH
Technische Universität Hamburg

WMI (ab Win2K) dient der Ereignis- und Leistungüber-wachung

der **PnP Manager** erkennt neue Geräte und fragt ggf. mit Hilfe des User-Mode Teils nach einem Treiber.

HAL ist die Hardware-Abstraktions-schicht

.inf und **.cat** Dateien kommen mit dem Treiber

der **E/A Manager** führt die Ein- und Ausgaben mit Hilfe der Treiber durch

Windows – Treiberstruktur

TUHH Technische Universität Hamburg

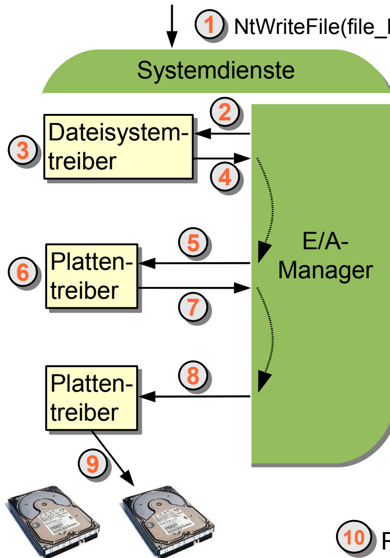
Das E/A-System steuert den Treiber mit Hilfe der ...

- Initialisierungsroutine/Entladeroutine
  - wird nach/vor dem Laden/Entladen des Treibers ausgeführt
- Routine zum Hinzufügen von Geräten
  - PnP Manager hat ein neues Gerät für den Treiber
- "Verteilerroutinen"
  - Öffnen, Schließen, Lesen, Schreiben und gerätespezifische Oper.
- Interrupt Service Routine
  - wird von der zentralen Interrupt-Verteilungsroutine aufgerufen
- DPC-Routine
  - "Epilog" der Unterbrechungsbehandlung
- E/A-Komplettierungs- und -Abbruchroutine
  - Informationen über den Ausgang weitergeleiteter E/A-Aufträge

...

Windows – typischer E/A-Ablauf

TUHH
Technische Universität Hamburg



1. über das Dateiobjekt wird das Dateisystem und der Treiber gefunden
2. Daten an bestimmten Byte-Offset in Datei schreiben
3. Position auf Datenträger berechnen
4. E/A Auftrag weitergeben
5. Daten an best. Byte Offset auf Datenträger schreiben
6. Position in Plattennr. und Offset umrechen
7. E/A Auftrag weitergeben
8. Daten an best. Byte Offset auf Platte 2 schreiben
9. Phys. Block berechnen und Operation initiieren

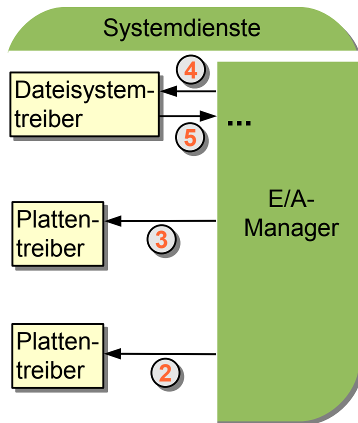
10 Rückkehr zum Anwendungsprozess!

Windows – typischer E/A-Ablauf

TUHH Technische Universität Hamburg

... Fortsetzung (nachdem die Platte fertig geworden ist)

1. Plattencontroller signalisiert per Unterbrechung den Abschluss der Operation
2. Aufruf der ISR bzw. des DPC
3. Aufruf der Komplettierungs-routine
4. Aufruf der Komplettierungs-routine
5. weiterer (Teil-)Auftrag an den Datenträgertreiber



Wo merkt sich das System den Zustand einer E/A-Operation?

Windows – E/A-Anforderungspaket

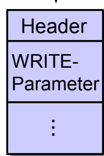
TUHH
Technische Universität Hamburg

NtWriteFile(file_handle, char_buffer)

Systemdienste

E/A-Manager

der E/A-Manager erstellt und initialisiert für jede E/A Operation ein IRP

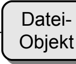


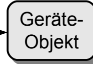
IRP
(I/O Request Packet)

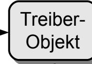
IRP-Stack

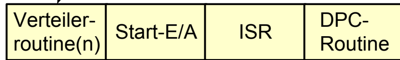
jede Treiberebene benutzt eine neue Ebene im IRP-Stack

über die WRITE-Parameter wird die Verteilerroutine gefunden









Gerätetreiber

Agenda

TUHH Technische Universität Hamburg

Einordnung
- Bedeutung von Gerätetreibern

Anforderungen an das BS
- Namensraum
- Einheitlicher Zugriff
- Spezifischer Zugriff

Struktur des E/A-Systems
- Treibermodell
- Linux
- Windows

Zusammenfassung

Zusammenfassung

- ein guter Entwurf des E/A Subsystems ist enorm wichtig
  - E/A-Schnittstelle
  - Treibermodell
  - Treiberinfrastruktur
  - Schnittstellen sollten lange stabil bleiben
- Ziel ist die Aufwandsminimierung bei der Treibererstellung
- Windows besitzt ein ausgereiftes E/A System
  - "alles ist ein Kern-Objekt"
  - asynchrone E/A Operationen sind die Basis
- Linux zieht rasant nach
  - "alles ist eine Datei"
  - sysfs und asynchrone E/A sind relativ neu (seit 2.6)