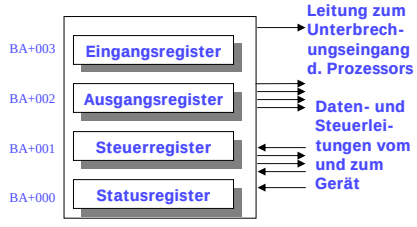
Kapitel 8

Ein- / Ausgabe

Treiber

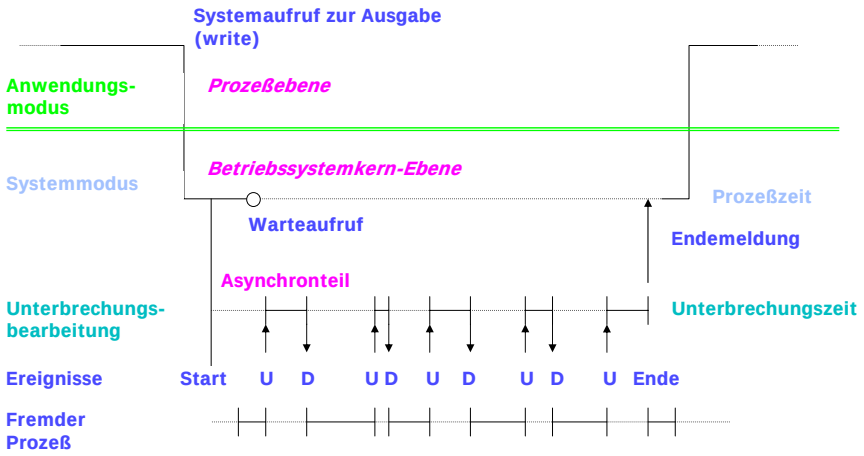
- **... stellen den direkten Kontakt zur Hardware her**
- **Anbindung von E/A-Geräten normalerweise nicht direkt über den Prozessor, sondern über gerätespezifische Steuerungen (Controller)**
- **Betriebssystemsicht: Gerät als ein Satz adressierbarer Baustein-Register sowie i.a. eine Leitung zum Unterbrechungseingang des Prozessors**
  - Eingangs-Register zur Übernahme von Daten an das Gerät vom Prozessor
  - Ausgangs-Register zur Übergabe von Daten vom Gerät an den Prozessor
  - Steuer-Register zur Einstellung von Bausteinparametern, Steuerung des aktuellen Datentransfers und Auslösung von einzelnen Aktionen
  - Status-Register zur Meldung des aktuellen Bausteinzustands
- **Kommandos**
  - werden bei manchen Geräten wie Daten transferiert
  - Bei anderen werden sie über eigene Status- und Steuer-Register geleitet



Synchroner und asynchroner Treiberteil
E/A-Ablauf

- **Der Prozeß übergibt einen Puffer mit Anfangsadresse und Länge an den Treiber**
- **Der Treiber kopiert ihn in einen betriebssystemeigenen Puffer um**
  - jederzeitiger Zugriff ist sichergestellt (auch bei Auslagerung auf die Platte)
- **Der Treiber gibt das erste Datum aus; dann legt er sich und damit den Prozeß schlafen (Schlafen im Treiber!) Bisher Ablauf zur Prozeßzeit; jetzt zur Unterbrechungszeit**
- **Das Gerät übernimmt ein Datum. Wenn es für das nächste Datum bereit ist, löst es eine Unterbrechung aus ("U" in der Abbildung).**
- **Die Unterbrechungsprozedur gibt das nächste Datum aus dem Puffer ans Gerät aus ("D" in der Abbildung). Dieser und der vorangehende Schritt wiederholen sich mehrfach.**
  (In der Zeit zwischen "D" und "U" kann ein fremder Prozeß den Prozessor bekommen, wird aber immer wieder unterbrochen)
- **Wenn der Puffer schließlich leer oder ein Fehler aufgetreten ist, weckt die Unterbrechungsprozedur den im Treiber schlafenden Prozeß und meldet ihm das Bearbeitungsende sowie - Verwaltungsinformationen wie z. B. Anzahl der übertragenen Daten oder einen Fehlercode**
- **Der geweckte Prozeß wird nun vom Betriebssystem bereit gesetzt, es kommt schließlich zum Prozeßwechsel, und der Rest des Treibers läuft wieder zur Prozeßzeit ab. Er kann nun noch einige Aufräumarbeiten durchführen (z. B. Puffer freigeben), bevor wieder auf die Anwendungsebene umgeschaltet und schließlich Code des Prozesses abgearbeitet wird.**

Synchroner und asynchroner Treiberteil (2)
E/A-Ablauf



Fallbeispiel: Treiber in Unix

- **Unix-Gerätekonzept: Geräte werden wie Dateien angesprochen**
  - Jedes Gerät kommt daher auch im Dateisystem vor, und zwar als Spezialdatei
  - Zusammenfassen aller Geräte im Katalog /dev und seinen Unterkatalogen
- **Eintrag im Dateisystem (Katalogeintrag und Inode) wird mit dem Systemaufruf bzw. Kommando mknod erzeugt**
  - Pfadnamen der Spezialdatei (also /dev/...),
  - Gerätetyp: Block- oder Zeichengerät (b / c),
  - Treiber- und Einheitennummer (major / minor number).
    - Major: Eintrag in der Konfigurationsdatei "master"; Index in Unix-interne Funktionstabellen
    - Minor: fortlaufende Einheitennummer, die vom gleichen Treiber bedient werden (5¼ / 3½")
- **Beispiel des mknod-Kommandos zur Einrichtung von Terminal Nr. 5:**
  - mknod /dev/tty05 c 3 5; ls -l

```
crw--w--w-   1 root     root       3,  5 Jan 11 14:27 tty05
```

- **Blockgeräte: Häufig zwei Treibernummern (Blockgerätzugriff/ Zeichengerätzugriff ("raw device")**

```
brw-rw-rw-   4 bin      bin        2, 52 Jan 12  1989 fd096ds15
crw-rw-rw-   4 bin      bin        2, 52 Jan 12  1989 rfd096ds15
c-w--w--w-   1 bin      bin        6,  1 Jan 12  1989 lp1
```


Konfigurationsdatei "master"

- **Konfigurationsdatei master enthält für jeden Gerätetyp (Treiber) ...**
  - Gerätenamen (max. 8 Zeichen)
  - Anzahl der belegten Unterbrechungsnummern
  - "Gerätemaske", die bitweise angibt, welche der Treiberfunktionen benötigt werden (also open / close, read / write usw.)
  - Funktionspräfix (Kurzform des Gerätenamens, unten mit xx bezeichnet)
  - Treibernummer für Blockgerätzugriff | (betreffen dasselbe Gerät)
  - Treibernummer für Zeichengerätzugriff |
  - spl-Ebene (1 bis 7) mit der Unterbrechungspriorität
  - und weitere Informationen
- **Die Unix-internen Funktionstabellen werden beim Konfigurationsvorgang automatisch aus den Einträgen in master erzeugt**
  - Schnittstellen zu den jeweiligen Treiberfunktionen
  - Anhand der Treibernummer wird bei Ansprechen eines Geräts (z. B. read) die entsprechende Gerätefunktion (z. B. ttyread) ausgewählt
  - beim Aufruf wird dann jener die Einheitennummer als Argument übergeben; Beispiel:

```
int fd;
   fd = open ("/dev/tty05", ...);  systemintern ==>   ttyopen (5, ...);
   read (fd, ...);                                    ttyread (5, ...);
```

Aufruf-Schnittstelle

- **Ein Treiber muß bestimmte Funktionen zur Verfügung stellen, die vom Unix-Kern bei Bedarf aufgerufen werden**
  - Mit "xx" ist eine Kurzform des jeweiligen Gerätenamens anzugeben
    hd (hard disk), fd (floppy disk), lp (line printer), tty (teletype), sio (serial i/o) für Festplatte, Diskettenlaufwerk, beliebigen Drucker, Terminal, serielle Schnittstelle
- **Geräteunabhängige Vorbereitungen (Beispiel open):**
  - Pfadnamen analysieren, übergebene Argumente prüfen, den Inode und dadurch das angesprochene Gerät bestimmen, den Eröffnungszähler inkrementieren und das Gerät in die Liste geöffneter Dateien/Geräte des aufrufenden Prozesses eintragen.
- **Geräteabhängige Aktionen**
  - Mittels der Funktionstabellen bdevsw oder cdevsw wird die geräteabhängige xxopen-Funktion bestimmt und schließlich mit der Einheitennummer als Argument aufgerufen

Zeichengeräte

- **Es müssen zumindest die folgenden Treiberfunktionen bereitgestellt werden, wobei von den zwei Funktionen xxintr und xxpoll genau eine benötigt wird**
  - xxinit zur Initialisierung des betreffenden Geräts beim Systemstart
  - xxopen, xxclose, xxread, xxwrite
  - xxioctl zur Parametrierung des Geräts/Schnittstelle und für gerätespezifische Aktionen
  - xxintr zur Unterbrechungsbehandlung
  - xxpoll zur periodischen Abfrage des Gerätestatus (falls keine Unterbrechungen)
  - xxstart zum Start einer E/A-Aktivität
- **Die Tabelle cdevsw (= character device switch table) hat etwa folgende Form:**

```
struct cdevsw cdevsw[] = {
/* 0*/ "scrn",    cnopen,  cnclose,  cnread,  cnwrite,  cnioctl,
/* 1*/ "hd",      hdopen,  nulldev,  hdread,  hdwrite,  hdioctl,
/* 2*/ "fd",      fdopen,  fdclose,  fdread,  fdwrite,  fdioctl,
/* 3*/ "tty",     ttyopen, ttyclose, ttyread, ttywrite, ttyioctl,
/* 4*/ "memory",  nulldev, nulldev,  mmread,  mmwrite,  nodev,
/* 5*/ "sio",     sioopen, sioclose, sioread, siowrite, sioioctl,
/* 6*/ "lp",      lpopen,  lpclose,  nodev,   lpwrite,  lpioctl,
/* 7*/ "rtc",     nulldev, nulldev,  rtcread, rtcwrite, nodev,
/* 8*/ 0,         nodev,   nodev,    nodev,   nodev,    nodev,
 ...
};
```

Zeichengeräte
Weitere Funktionstabellen

- **Weitere Funktionen**
  - werden innerhalb des Treibers, jedoch nicht vom Unix-Kern aufgerufen
  - z. B. xxstart zum Auslösen einer E/A-Aktivität, wenn das Gerät untätig ist
  - Gewöhnlich reagiert das Gerät darauf später mit einer Unterbrechungsanforderung
- **Andere Funktionen werden in eigenen Funktionstabellen gemeinsam für beide Gerätetypen geführt**
  - Funktion xxinit über die Tabelle dinitsw (= device init switch table)
  - xxpoll über dpollsw (= device poll switch table)
  - Tabelle vecintsw (= device vector interrupt switch table) enthält gerätespezifische Unterbrechungsprozeduren

Blockgeräte

- **Es müssen prinzipiell folgende Treiberfunktionen bereitgestellt werden:**
  - xxinit zur Initialisierung des betreffenden Geräts beim Systemstart,
  - xxopen, xxclose,
  - xxintr zur Unterbrechungsbehandlung,
  - xxstrategy zur Einordnung eines Transferauftrags in die Auftragsliste,
  - xxstart zum Start einer E/A-Aktivität (Lesen oder Schreiben).
- **Die folgenden drei Funktionen werden nur für den Zugriff über das zugehörige raw device benötigt:**
  - xxread
  - xxwrite
  - xxioctl
- **Die Funktionstabelle bdevsw (= block device switch table)...**

```
struct bdevsw bdevsw[] = {
/* 0*/ 0,    nodev,  nodev,   nodev,      0,
/* 1*/ "hd", hdopen, nulldev, hdstrategy, &hdtab,
/* 2*/ "fd", fdopen, fdclose, fdstrategy, &fdtab,
/* 3*/ 0,    nodev,  nodev,   nodev,      0,
 ...
};
```

- **xxstrategy -- entweder disksort oder selbst implementieren**

Blockgeräte
Weitere Funktionen

- **xxstrategy ruft xxstart zum Anstoß des eigentlichen E/A-Vorgangs auf**
- **xxstart berechnet für den ersten Auftrag in der Warteschlange die physikalische Adresse auf der Platte und schickt den Auftrag an die Festplattensteuerung, und zwar gleichermaßen zum Lesen und Schreiben**
- **Die Steuerung antwortet darauf später irgendwann mit einer Unterbrechungs-anforderung, die zum Aufruf von xxintr führt**
- **Die Abarbeitung weiterer Aufträge in der Warteschlange wird ebenfalls mittels xxstart von xxintr aus angestoßen, bis alle Aufträge abgearbeitet sind**

- **Die Funktionen xxread und xxwrite werden nur für den ungepufferten Zugriff über das zum Blockgerät gehörige zeichenorientierte raw device gebraucht. Sie rufen ihrerseits die Funktionen xxstrategy und damit xxstart auf.**
- **xxioctl löst gerätespezifische Aktionen aus wie z. B. einen Formatiervorgang, Parken der Festplattenköpfe, Öffnen und Schließen der Laufwerkstür, Auswerfen des Datenträgers, Zurückspulen eines Magnetbandes usw.**

Hilfsfunktionen des Unix-Kerns

- **Hilfsfunktionen des Unix-Kerns**
  - zur kerninternen Speicherverwaltung
  - zur Umsetzung von logischen in physikalische Adressen und umgekehrt
  - für den Datentransfer in und aus dem Adreßraum des Anwendungsprozesses
  - für den DMA-Betrieb
  - zur Block- und Zeichenpufferung
  - zur Ausgabe von Fehler- und Testmeldungen auf der Systemkonsole

!!! Alle treibereigenen Daten befinden sich im kerninternen Adreßraum !!!

- **spl()-Funktionen (spl = set priority level)**
  - Sperren von Unterbrechungen auf sieben Ebenen
- **Koordinierungsfunktionen sleep, wakeup**
  - mit sleep(kanal, sysprio) kann sich ein Treiber im Synchronteil schlafen legen und auf ein bestimmtes Ereignis warten (Prozeßzustand "wartend").
  - Er wird dann vom Aufruf wakeup(kanal) (üblicherweise aus der Unterbrechungsbehand-lung heraus) geweckt, also bereit gesetzt, und kann dann mit der Systempriorität sysprio weiter arbeiten
  - timeout: Hiermit wird nach einer angegebenen Anzahl von Ticks eine im Aufruf spezifizierte Funktion aufgerufen (z.B. für erneutes HW-Ansprechen oder timeout)

Datenstrukturen des Unix-Kerns

- **clists (= character lists)**
  - Gut geeignet für Datenaustausch zwischen Synchron- und Asynchronteil des Treibers
  - Eine clist ist eine treibereigene, verzeigerte Liste von cblocks.
    - Ein cblock enthält variable Anzahl von Zeichen (bis zu einem Maximum von ca. 32)
    - einen Zeiger auf den Nachfolgerblock
    - je einen Zeiger auf das erste und letzte gültige Zeichen im Block
    - Systemweiter Vorrat an cblocks
    - Hilfsfunktionen cpass() und passc() sorgen für den Zeichentransfer
    - Zur clist gehört eine Füllstandsvariable mit der Anzahl der Zeichen in der Liste
  - Bei Überschreiten einer "Hochwassermarke", wird der ausgebende Prozeß mittels der sleep-Funktion blockiert, bis die "Niedrigwassermarke" wieder unterschritten ist
- **Blockpuffer - Pufferung von Transferaufträgen**
  - Kerneigener Vorrat an freien Blöcken (Nutzdatenblock der Größe BSIZE (= 512 Bytes oder Vielfaches) und ein Vorspann (header)
    - Treiber- und Einheitennummer des zugehörigen Geräts
    - (logische) Blocknummer auf dem Gerät
    - physikalische Hauptspeicher-Adresse des zugehörigen Nutzdatenblocks
    - Anzahl der zu transferierenden Bytes
    - Statuswort mit Frei- und Fehlerindikator sowie "Verändert"-Bit
    - Vorwärts- und Rückwärtszeiger für Freiliste und Auftragswarteschlange

Datenstrukturen des Unix-Kerns (2)

- **Der Unix-Kern ...**
  - verwaltet den Vorrat an freien Pufferblöcken
  - führt für belegte Pufferblöcke Buch darüber, von welcher Datei er Daten enthält und ob er verändert wurde (und daher auf die Platte geschrieben werden muß)
- **Bei read-Aufträgen ...**
  - Unix-Kern prüft, ob die gewünschten Daten schon im Hauptspeicher vorhanden sind und veranlaßt den Treiber, nur noch die fehlenden Blöcke zu lesen.
- **Bei write-Aufträgen ...**
  - Unix-Kern kopiert die Daten in Pufferblöcke
  - Wenn nicht mehr genügend freie Blöcke vorhanden sind, müssen einige der veränderten Blöcke auf die Platte geschrieben werden, um Platz zu schaffen
  - Wenn alle zu transferierenden Daten in Pufferblöcke kopiert wurden, kann der Prozeß fortgesetzt werden (verzögertes Schreiben).

Datenstrukturen des Unix-Kerns (3)

- **tty-Struktur, u.a. für die Beschreibung eines Terminals ("tty")**
  - die Listenanker dreier clists
  - eine Reihe von Status- und Steuerwörtern zur Festlegung bestimmter Betriebsarten
  - die Definition von Zeichen mit Spezialbedeutung (z. B. das Dateiendezeichen, Eingabebegrenzungszeichen wie Zeilenende, Steuerzeichen für den Prozeßabbruch)
  - Prozeßgruppennummer der zugeordneten Prozesse
  - Vorschriften zur Behandlung bestimmter Sonderzeichen (Tabulator-, Löschzeichen usw.) und deren Definition
  - die Angabe, ob Groß- in Kleinbuchstaben umgewandelt werden sollen
  - zeitliche Parameter und Ausgabegeschwindigkeiten
  - ggf. Parameter für die serielle Schnittstelle
  - und vieles anderes mehr.
- **Prozeßbezogene user-Struktur (~ Prozeßkopf)**
  - u_segflag, gibt an, ob es sich bei den zu transferierenden Daten um Systemdaten, Anwendungsdaten oder Anwendungscode handelt,
  - u_base mit der Anfangsadresse des zu transferierenden Datenblocks im Adreßraum des Prozesses,
  - u_count mit der aktuellen Zahl der noch zu transferierenden Bytes und
  - u_offset mit der Transferposition in der Datei (bzw. dem Gerät)