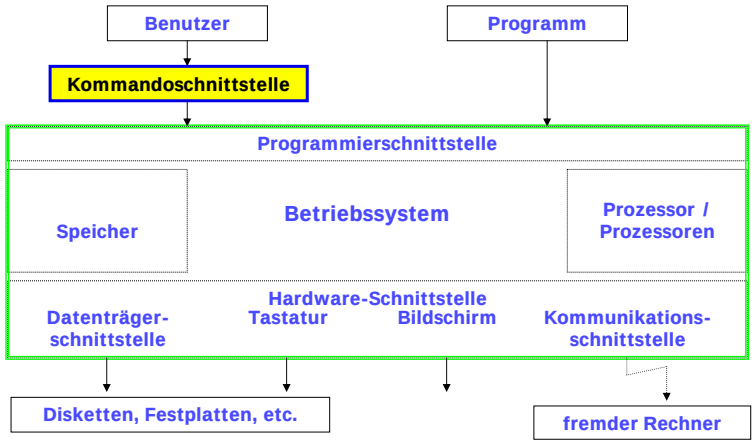
Kapitel 2

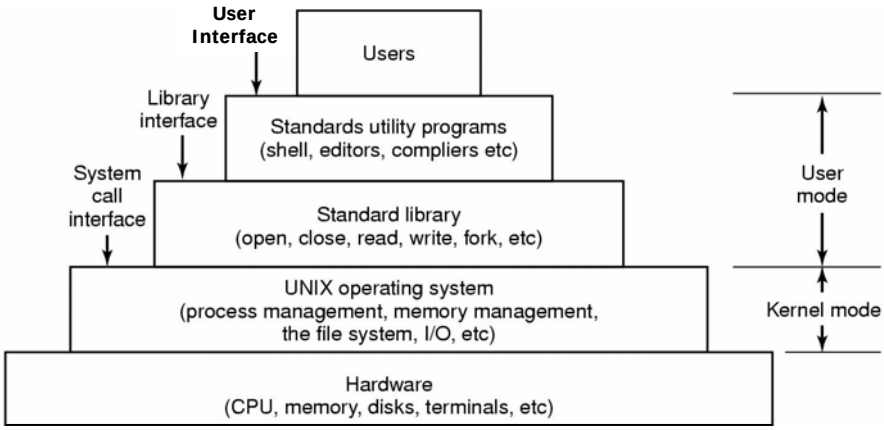
Kommando- und Programmierschnittstelle

Unix – Shell
MS-DOS Programmierschnittstelle auf Assemblerebene

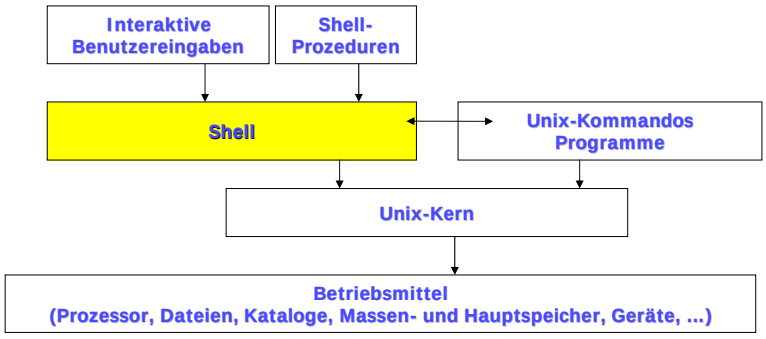
Architektur von Betriebssystemen



Struktur des UNIX Betriebssystems



Die Stellung der Shell und Standarddateien in Unix



- Standard-Eingabe *(stdin)* z.B. Tastatur
- Standard-Ausgabe *(stdout)* z.B. Bildschirm
- Standard-Fehlerausgabe *(stderr)* z.B. Bildschirm

Eigenschaften der UNIX Shell (Kommandozeilen-Schnittstelle)

- **Keine graphische Benutzerschnittstelle**
- **Die Shell ist ein Programm (Interpreter), das ständig auf Eingaben wartet und diese direkt verarbeitet**
- **Mehrere Varianten**
  - UNIX Bourne Shell (sh)
  - C-Shell, TC-Shell (csh, tcsh)
  - Bash (siehe Linux), Korn-Shell
- **Eigenschaften**
  - Mächtig
  - Einfach erweiterbar
  - Schnell bedienbar
- **➔ Shell-Programme und Shell-Skripten sind mächtige Werkzeuge zur Behandlung von Routine-Tätigkeiten**

Arbeiten mit der Unix Shell

- **Shell wird beim Anmelden des Benutzers als eigener Prozess gestartet**
- **Benutzereingabe (Beispiel) : date**
  - Shell erzeugt einen weiteren Prozess, der das Programm date ausführt
  - Während der Prozess ausgeführt wird, wartet der Prozess auf dessen Ende
  - Wenn der Prozess beendet ist, ist die Shell erneut eingabe-bereit und wartet auf folgende Kommandos

UNIX-Programme mit Verwendung über die Kommandozeile

Program	Typical use
cat	Concatenate multiple files to standard output
chmod	Change file protection mode
cp	Copy one or more files
cut	Cut columns of text from a file
grep	Search a file for some pattern
head	Extract the first lines of a file
ls	List directory
make	Compile files to build a binary
mkdir	Make a directory
od	Octal dump a file
paste	Paste columns of text into a file
pr	Format a file for printing
rm	Remove one or more files
rmdir	Remove a directory
sort	Sort a file of lines alphabetically
tail	Extract the last lines of a file
tr	Translate between character sets

Programme und Kommandos der UNIX Shell

- **cp quelle ziel**
  - Ruft das cp UNIX-Kommando mit den zwei Argumenten quelle und ziel auf
  - Interpretiert die beiden Argumente als Dateinamen
  - Kopiert Inhalt der ersten existierenden Datei in die zweite Datei
- **head -20 datei**
  - Gibt die ersten 20 Zeilen der Datei datei auf der Standardausgabe aus
  - (standardmäßig würde head nur die ersten 10 Zeilen ausgeben)
  - Argumente, welche die Operation eines Kommandos beeinflussen oder optionale Werte spezifizieren, werden Optionen genannt und beginnen mit einem ‚-' !
- **head 20 datei**
  - Gibt die ersten 10 Zeilen der beiden Dateien namens 20 und datei auf der Standardausgabe aus

UNIX Shell – Ein/Ausgabe-Umlenkung

- **sort**
  - Liest Zeichen von Standard-Eingabe (i.Allg. vom Terminal/Tastatur), sortiert diese und gibt sie auf der Standard-Ausgabe (i.Allg. auf dem Bildschirm) wieder aus
- **sort < in > aus**
  - Liest Eingaben aus der Datei in, sortiert sie und schreibt sie in die Datei aus
  - Umlenkung von Standard-Ausgabe und Standard-Eingabe
- **sort < in > tmp; head -30 tmp; rm tmp**
  - Liest Eingaben aus der Datei in, sortiert sie und schreibt sie in die Datei tmp
  - Gibt die ersten 30 Zeilen der Datei tmp auf der Standard-Ausgabe aus
  - Löscht die Datei tmp

UNIX Shell – Ein/Ausgabe-Umlenkung

Redirection Operators

- **<**
  Leitet die Standard-Eingabe um
- **>**
  Leitet die Standard-Ausgabe um
- **>>**
  Hängt die Standard-Ausgabe an (i. Allg. bereits bestehende) Datei an

Mehrfachnamen (wildcards)

- **Nicht eindeutige Mehrfachnamen (Namensmuster, wildcards)**
  - * steht für eine beliebige (auch leere) Zeichenkette,
  - ? steht für genau ein beliebiges Zeichen,
  - [...] steht für genau eines der in der Klammer enthaltenen Zeichen. Ein durch "-" getrenntes Zeichenpaar steht für genau eines derjenigen Zeichen, die lexikographisch zwischen den beiden Zeichen des Paares liegen.
- **Beispiele**
  - *tmp*
  - ?/tmp*
  - [12a-e]??
  - *.c
  - .*
- **„Zitieren“ von Metazeichen: Voranstellen von „\“**
  - **
  - \\a\?

FOR-Schleife und Übergabeparameter

```
for C_PROG in prog1 prog2 prog3        | Wirkung: cc -o prog1 prog1.c
do                                     |          cc -o prog2 prog2.c
   cc -o $C_PROG $C_PROG.c             |          cc -o prog3 prog3.c
done
```

- **Leere FOR-Parameterliste**

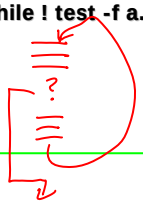
```
for WORT                +
do                       >  echo2
   echo $WORT$WORT       >
done                    +
```

- **echo2 ma pa fi; echo2 "ma pa fi"**
  - mama
  - papa
  - fifi
  - ma pa fima pa fi

while / until-Schleife

- while <liste1>; do <liste2>; done
- until <liste1>; do <liste2>; done

```
# test -f <name> testet, ob die Datei namens <name> existiert.
(while test ! -f a.out    : oder: until test -f a.out   oder: while ! test -f a.out
do
  sleep 30
done
echo ^G Uebersetzung ist fertig!)&
```




```
(while true            : true liefert immer 0
do
  date
  echo ^G
  sleep 1800
done) &
```


if-Kommando

```
if <liste1>
then <liste2>
[elif <liste3>
then <liste4> ]
 ...
[else <liste5>]
fi
```

```
if   test "$VAR1" = "$VAR2"
then echo Die Variablen sind gleich!
fi
```

```
if test -f $DATEI; then cat $DATEI; else echo Datei $DATEI nicht vorhanden; fi
```

case-Kommando

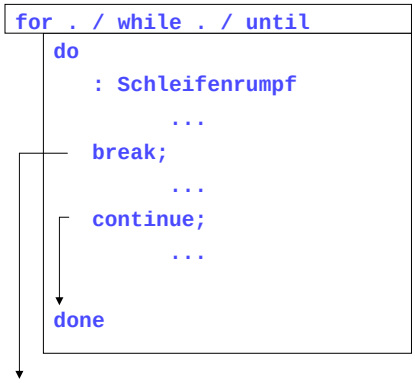
```
case <wort> in [<muster> [|<muster>] ... ) <liste> ;;] ...
esac
```

```
case $DATEI in
    *.c)          cc $DATEI; echo $DATEI wurde uebersetzt ;;
    *.h | *.sh)   : tu nichts ;;
    *)            echo Dateityp von $DATEI unbekannt ;;
esac
.. ;; beendet hier auch den Kommentar!
```


Test-Kommando

- **Optionen mit Dateien und Directories**
  - -r <datei> wahr, wenn <datei> existiert und für den Benutzer lesbar ist.
  - -w <datei> wahr, wenn <datei> existiert u. für den Benutzer beschreibbar ist.
  - -x <datei> wahr, wenn <datei> existiert und für den Benutzer ausführbar ist.
  - -d <datei> wahr, wenn <datei> ein Katalog ist.
  - -f <datei> wahr, wenn <datei> existiert und eine gewöhnliche Datei ist.
  - -s <datei> wahr, wenn <datei> existiert und eine Länge > 0 hat.
- **Einige Optionen mit Zeichenketten**
  - -z <zk1> wahr, wenn die Zeichenkette <zk1> die Länge 0 hat.
  - -n <zk1> wahr, wenn die Zeichenkette <zk1> eine Länge > 0 hat.
  - <zk1> = <zk2> wahr, wenn die Zeichenketten <zk1> und <zk2> identisch sind.
  - <zk1> != <zk2> wahr, wenn <zk1> und <zk2> nicht identisch sind.
- **Optionen mit ganzen Dezimalzahlen**
  - <n1> -eq <n2> wahr, wenn die Zahlen <n1> und <n2> algebraisch gleich sind.
  - <n1> -ne <n2> wahr, wenn die Zahlen <n1> und <n2> algebraisch ungleich sind.
  - <n1> -gt <n2> wahr, wenn <n1> > <n2> gilt.
  - <n1> -ge <n2> wahr, wenn <n1> => <n2> gilt.
  - <n1> -lt <n2> wahr, wenn <n1> < <n2> gilt.
  - <n1> -le <n2> wahr, wenn <n1> <= <n2> gilt.

break / continue / exit



Variablen und Parameter

- **Variable**
  - ABC=k1; ABCD=k2 !!! Keine Leerzeichen zwischen Var. und = !!!
  - Z=${ABC}D : Z hat den Wert k1D
  - Z="$ABC"D : Z hat den Wert k1D
  - Z=$ABCD : Z hat den Wert k2
  - Z='$ABC'D : Z hat den Wert $ABCD

- **Positionsparameter**
  - echo2 ma "pa pa“ fi
  - haben die vier Positionsparameter $0 bis $4 die Werte
  - $0: echo2, $1: ma, $2: pa pa, $3: fi ; $4 ist leer
  - Der *shift-Befehl* schiebt die Parameter ab $2 um eine Position nach links!
  - $0 $2 $3 $4 $5 ... in der Numerierung $0 $1 $2 $3 $4 ...

Spezielle Parameter

- **#** die Anzahl der Positionsparameter in einem Aufruf, jedoch ohne $0
- **"$*"** ist äquivalent zu "$1 $2 ... ", ergibt bei Weitergabe nur einen Positionsparameter
- **"$@"** ist äquivalent zu "$1" "$2" ..., ergibt die ursprüngliche Anzahl der Pos.Parameter
- **?** der Wert des letzten ausgeführten Kommandos (oder Programms).
- **$** die Prozeßnummer der Shell (dient z. B. zur Benennung temporärer Dateien)
- **!** die Prozeßnummer des letzten Hintergrundkommandos (Anwendung z. B. in kill $!).

- **HOME** Name des Heimat-Katalogs (z. B. /usr/kern).
- **PATH** Suchpfad für ausführbare Dateien, z. B.: :/bin:/usr/bin:/$HOME/bin
- **MAIL** Name der "Briefkasten-Datei", in der angekommene Post vermerkt wird
- **PS** primary prompt string, normalerweise: '$ '.
- **PS2** secondary prompt string, normalerweise: '> '.
  - Er erscheint, wenn ein strukturiertes Kommando (z. B. for-Schleife) bei Eingabe "von Hand" noch nicht abgeschlossen ist.
- **IFS** internal field separators, legt die zulässigen Trennzeichen fest, normalerweise: <Leerzeichen>\t\n.

read-Kommando

- **read <variable> {<variable>}**
  - Der Wert von read ist 0, solange kein EOF (End of File, = ^D) gelesen wurde

- **Beispiel:**
  - read VAR1 VAR2 VAR3

  Die Eingabezeile für dieses Kommando:
  - Dies ist eine ganz lange Zeile.

  - **echo $VAR1 : Dies**
  - **echo $VAR2 : ist**
  - **echo $VAR3 : eine ganz lange Zeile**

Shell-Arithmetik: expr

Arithmetische und logische Operationen auf ganzen Zahlen und Zeichenketten

- **!!! Leerzeichen zwischen Operanden und Operatoren !!!**
- **!!! Quotieren des Multiplikationsoperators "*" !!!**
- **!!! Verwenden des ` ` - Befehls bei der Zuweisung der Ergebnisse !!!**

  - SUMME=`expr $SMD1 + $SMD2 + 1`
  - PRODUKT=`expr $FAKTOR1 \* $FAKTOR2`

  - A=3 B=9
  - expr $A * $B : bewirkt die Ausgabe: Syntax error
  - expr $A * $B : bewirkt die Ausgabe: 27
  - expr $A*$B : bewirkt die Ausgabe: 3*9
  - expr $A * \(5 + $B \) : bewirkt die Ausgabe: 42

- **arithmetische Operatoren: + - * / %**
- **Vergleichsoperatoren: < <= = != >= >**

copyto

```
: Benutzung: copyto kat datei ...
: kopiert Argumentdateien nach "kat"
: stellt vorher sicher, daß mindestens zwei Argumente existieren
: und daß "kat" ein Katalog ist

if test $# -lt 2
then    echo "$0: Syntax: $0 katalog datei ..."
elif test ! -d $1
then    echo "$0: $1 ist kein Katalog"
else kat=$1; shift
    for datei
    do
      cp $datei $kat   : cp: Normaler Kopierbefehl
    done
fi
```

fsplit

```
: Gebrauch: fsplit datei1 datei2
: Liest stdin und teilt sie zeilenweise in drei Teile auf:
: Jede Zeile, die mindestens einen Buchstaben enthält, wird an datei1 angehängt;
: jede Zeile, die mindestens eine Ziffer, aber keinen Buchstaben enthält, wird an datei2 angehängt;
: der Rest wird verworfen.

zaehler=0;  ausschuss=0

while read zeile
do
  zaehler=`expr $zaehler + 1`
  case $zeile in
  *[A-Za-z]*)     echo $zeile >> $1 ;;
  *[0-9]*)        echo $zeile >> $2 ;;
  *)              ausschuss=`expr $ausschuss + 1`
  esac
done

echo $zaehler Zeilen gelesen, davon $ausschuss Zeilen verworfen
```


wc-Befehl, interaktiv

```
echo -n "   Was soll gezaehlt werden?
   1   Anzahl der Zeichen
   2   Anzahl der Woerter
   3   Anzahl der Zeilen
   Bitte geben Sie die entsprechende Ziffer ein!"

read Ziffer
case $Ziffer in
   1)  Option=c   ;;
   2)  Option=w   ;;
   3)  Option=l   ;;
   *)  echo Ungueltige Eingabe; exit 1  ;;
esac

echo -n "   Welche Datei soll gezaehlt werden?
   Bitte geben Sie den Namen ein!"
read Name
wc -$Option $Name
```

Umwandlung der Datumsanzeige

- Tue Oct 30 09:11:22 GMT 1990 --> Dienstag, 30. Oktober 1990, 09:11:22 Uhr GMT

```
set `date` ;   : Damit werden die Positionsparameter $i mit den Komponenten
               : der Datumsausgabe besetzt
case $1 in
  Mon)         Wochentag=Montag ;;
  Tue)         Wochentag=Dienstag ;;
  :
  Sun)         Wochentag=Sonntag ;;
esac

case $2 in
  Jan)         Monat=Januar ;;
  Feb)         Monat=Februar ;;
    :
  Nov)         Monat=November ;;
  Dec)         Monat=Dezember ;;
esac

echo $Wochentag, $3. $Monat $6, $4 Uhr $5
```


Weitere Hilfsmittel der UNIX Shell

- **Suche nach Zeichenketten**
  - grep –e “search term“ filename
- **Verknüpfen von Kommandos über Pipes (Pipelines)**
  - grep –e “search term“ filename | sort | head -10 | tail -5 > out.txt
  - Keine Zwischenspeicherung in temporären Dateien erforderlich
- **Bearbeitung im Hintergrund**
  - Shell wird nicht blockiert
  - Beispiel: (grep –e “search term“ filename | sort ; echo READY) &
- **Hilfe zu Shell Kommandos**
  - man commando_name

Weitere Hilfsmittel der UNIX Shell

- **Gruppierung von Kommandos**
  - (cmd1; cmd2 | grep –e “search“) | wc
- **Aliasing**
  - alias my_ls=“ls –al“
  - Definition eigener häufig verwendeter Komandos
- **Nicht-interaktiver Editor**
  - sed –e “s/term_old/term_new/g “ filename
  - Stream Editor
  - Bearbeitung von Dateien, die für interaktive Editoren zu groß sind, oder wenn gleiche Änderungen in mehreren Dateien vorgenommen werden sollen
- **Interaktive Editoren**
  - Emacs
  - vim : http://www.vim.org

Weitere Hilfsmittel der UNIX Shell

- **Weitere zahlreiche Hilfsmittel und Kommandos**
- **z.B. zur Verwaltung der Sicherheit**
  - chmod go-rwx filename
  - chmod 750 filename
  - chown new_owner filename
  - chgrp new_grp filename
- **Allgemeines UNIX Konzept**
  - Viele kleine, sehr flexible Werkzeuge
  - Kombination dieser Werkzeuge zu komplexen Kommandos und Programmen
  - Interaktive Benutzung oder „Shell-Programmierung“

Interaktive Shell-Benutzung oder Shell-Programmierung ?

	Einmalige Verwendung	Häufige Verwendung
Komplexe und umfangreiche Aktivität	Programm	Programm
Einfache Aktivität	Interaktiv	Programm

Arbeiten mit der UNIX-Shell

- Linux auf den PCs im Fachbereich Informatik der FH Nürnberg
- Linux-ähnliche Umgebung auf Windows PCs:
  http://www.cygwin.com
  - Shell, vim, etc.

Exkurs Rechner-Architektur

Beispiel für eine Programmierschnittstelle auf Assembler-Ebene

Exkurs Rechner-Architektur
Assemblerprogrammierung, Systemmodus, *Software-Interrupt*

- Prozessor führt nur Befehle einer Maschinensprache in Binärform aus
- Liegen die Maschinenbefehle in für Menschen lesbarer Form vor, so spricht man von der Assemblersprache des Prozessors
- Jeder Prozessortyp hat seine eigene Assembler- bzw. Maschinensprache
- Assemblerprogrammierung
  - keine Variablen mit Datentypen und Konstrukte zur Ablaufsteuerung (z. B. *for*- oder *while*-Schleifen)
  - Typlose Speicherstellen, Prozessorregister und (bedingte) Sprungbefehle
- Leistungsfähige Prozessoren können in (mindestens) zwei Zuständen arbeiten:
  - Systemzustand (privilegierter Modus, *system / privileged / kernel mode*)
    - Ausführung aller Maschinenbefehle
    - Ansprechen aller Prozessorregister, Speicher- und Geräteadressen

    Damit ist die gesamte Hardware frei zugänglich. Aus Sicherheitsgründen ist diese Betriebsart dem Betriebssystem vorbehalten (daher auch *kernel mode*).
  - Anwendungszustand (Normalmodus, *normal / user / application mode*)

Exkurs Rechner-Architektur - 2
Assemblerprogrammierung, Systemmodus, *Software-Interrupt*

- **Diese Unterscheidung von zwei Prozessorzuständen erlaubt es, sichere Betriebs-/ Rechnersysteme zu konstruieren, da einem normalen Applikationsprogramm sicherheitskritische Aktionen verwehrt sind**
  - **Wechsel aus der System- in die Anwendungsbetriebsart ist jederzeit problemlos möglich**
  - **umgekehrt jedoch nur durch spezielle Maschinenbefehle (z. B. *SVC / Supervisor Call* oder *Software-Interrupts*)**

    Sie ändern nicht nur den Prozessorzustand, sondern springen außerdem an eine in der Anwendungsbetriebsart weder sichtbare noch manipuliere Adresse, den Einsprungpunkt des Betriebssystems, das von diesem Punkt an die Kontrolle über das weitere Geschehen übernimmt, insbesondere über die Ausführung nun erlaubter privilegierter Maschinenbefehle wacht!
- **Software-Interrupt:**
  - **indirekter Unterprogrammsprung mit Übergang aus der Normal- in die Systembetriebsart**
  - **gleichzeitig Sichern des aktuellen Prozessorzustands, damit dieser später beim Rücksprung wiederhergestellt werden kann**

Exkurs Rechner-Architektur - 3
Assemblerprogrammierung, API

- **Die Schnittstelle eines Betriebssystems**
  - **Programmier- oder Systemaufruf-Schnittstelle *(system call interface, application program interface, API)***
  - **Diese Schnittstelle wird jedoch durch Hochsprachen für den Programmierer verdeckt**
- **Beispiel: Datenausgabe auf dem Bildschirm**
  - **Pascal: *WRITE*- oder in C: *printf*-Aufruf**
  - **Assemblerprogrammierer: Verwendung der sog. Systemaufrufe (*system* oder *supervisor calls*)**
  - **Bildschirmausgabe einer durch ein $-Zeichen begrenzten Zeichenkette (Prozessor: Intel 8086 unter MS-DOS)**

```
MOV DX,offset pufadr    ; DX, AH: Prozessor-Register
                        ; MOV: reiner Datentransfer (move)
MOV AH, 09              ; 9: Funktionsnummer für Bildschirmausgabe
INT 21H                 ; MS-DOS-Systemaufruf
...                     ; hier geht's nach dem Systemaufruf weiter
```

Exkurs Rechner-Architektur – 4
Programmierschnittstelle, INT 21H; CP/M- und Unix-Form

- **Bildschirmausgabe einer durch ein $-Zeichen begrenzten Zeichenkette**

CP/M Form:

```
MOV DX, offset pufadr    ; DX, AH: Prozessor-Register
MOV AH, 09               ; 9: Funktionsnummer für Bildschirm-Ausgabe
INT 21H                  ; MS-DOS-Systemaufruf
```

Unix Form:

```
LDS DX, pufadr           ; lädt DS- und DX-Register des 80x86 mit
                           Pufferadresse
MOV CX, anzahl           ; lädt CX-Register mit Pufferlänge
MOV BX, !                ; Standard-Ausgabegerät stdout (CON)
MOV AH, 40H              ; Funktionsnummer für Ausgabe
INT 21H                  ; MS-DOS-Systemaufruf

  BX = 1                 ; stdout (CON)
  BX = 4                 ; PRN
```


Exkurs Rechner-Architektur - 5
Assemblerprogrammierung, Software-Interrupt

- **INT 21H: Software-Interrupt-Befehl Nr. 21H (21H = 33)**
  - MS-DOS führt allerdings keine Modus-Umschaltung aus dem Anwendungsmodus in den privilegierten Systemmodus durch (wurde vom 8086 nicht unterstützt)
  - Unter MS-DOS arbeitet der Prozessor also immer im privilegierten Modus!

- **Systemaufrufe werden grundsätzlich mit einer Erfolgs- bzw. Fehlermeldung beendet; Varianten:**
  - Erfolg: Rückgabewert ≥ 0, Fehler < 0
  - Erfolg: Rückgabewert = 0, sonst Fehler
  - evtl. gesetztes Übertragsbit des Prozessors (Carry Flag)

- **Reaktionsmöglichkeiten / Notwendigkeit auf Fehler**
  Beispiel: Öffnen einer Datei
  - Fehlermöglichkeiten ...?

Exkurs Rechner-Architektur - 6
Assemblerprogrammierung, Ursachen für Fehlerfälle

- **Hardwareprobleme**
  typisch für Systemaufrufe (BS bildet Schnittstelle zur Hardware)
  - Diskette nicht eingelegt
  - Zugriff auf defekte Plattensektoren
  - Netzlaufwerk nicht vorhanden (keine Verbindung)
- **Unzureichende Rechte für die gewünschte Operation**
  typisch für Mehrprozeß- und Mehrbenutzer-Betriebssysteme
  - Schreibzugriff auf eine schreibgeschützte Datei
  - Zugriff auf einen anderweitig reservierten Drucker
  - Start eines Programms, für das man keine Ausführungsrechte besitzt
  - Zugriff auf einen geschützten Speicherbereich).
- **Falsche Parameterversorgung des Systemaufrufs**
  typisch für modularisierte Programm-Systeme mit Schnittstellen zwischen Modulen
  - ungültige Aufrufnummer
  - Überschreitung von vorgegebenen Grenzwerten