

Überblick: Teil D Betriebssystemabstraktionen

15 Programmausführung, Nebenläufigkeit

16 Ergänzungen zur Einführung in C

17 Betriebssysteme I

18 Dateisysteme

19 Prozesse I

20 Speicherorganisation

21 Prozesse II

22 Betriebssysteme II

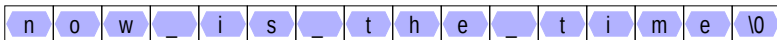


Zeiger, Felder und Zeichenketten

- Zeichenketten sind Felder von Einzelzeichen (**char**), die in der internen Darstellung durch ein `'\0'`-Zeichen abgeschlossen sind

- wird eine Zeichenkette zur Initialisierung eines char-Feldes verwendet, ist der Feldname ein konstanter Zeiger auf den Anfang der Zeichenkette

```
char amessage[] = "now is the time";
```



amessage $\equiv$

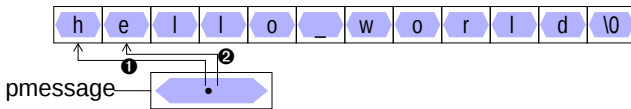
- es wird ein Speicherbereich für 16 Bytes reserviert und die Zeichen werden in diesen Speicherbereich hineinkopiert
- amessage ist ein konstanter Zeiger auf den Anfang des Speicherbereichs und kann nicht verändert werden
- der Inhalt des Speicherbereichs kann aber modifiziert werden
`amessage[0] = 'h';`



... Zeiger, Felder und Zeichenketten (2)

- wird eine Zeichenkette zur Initialisierung eines **char**-Zeigers verwendet, ist der Zeiger eine Variable, die mit der Anfangsadresse der Zeichenkette initialisiert wird

```
char *pmessage = "hello world";
```



```
pmessage++; ②  
printf("%s", pmessage); /* gibt "ello world" aus */
```

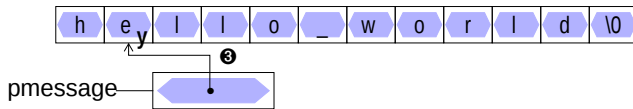
- es wird ein Speicherbereich für einen Zeiger reserviert (z. B. 4 Byte) und der Compiler legt die Zeichenkette hello world an irgendeiner Adresse im Speicher des Programms ab
- pmessage ist ein variabler Zeiger, der mit dieser Adresse initialisiert wird, aber jederzeit verändert werden darf
pmessage++;



... Zeiger, Felder und Zeichenketten (3)

- wird eine Zeichenkette zur Initialisierung eines **char**-Zeigers verwendet, ist der Zeiger eine Variable, die mit der Anfangsadresse der Zeichenkette initialisiert wird

```
char *pmessage = "hello world";
```



```
*pmessage = 'y'; ③
```



- der Speicherbereich von `hello world` darf aber nicht verändert werden
 - manche Compiler legen solche Zeichenketten in schreibgeschütztem Speicher an
 - ➔ Speicherschutzverletzung beim Zugriff
 - sonst funktioniert der Zugriff obwohl er nicht erlaubt ist
 - ➔ Programm funktioniert nur in manchen Umgebungen

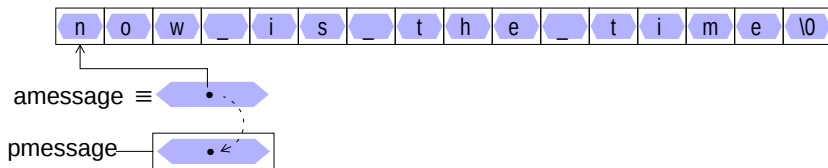


... Zeiger, Felder und Zeichenketten (4)

- die Zuweisung eines **char**-Zeigers oder einer Zeichenkette an einen **char**-Zeiger bewirkt kein Kopieren von Zeichenketten!

```
pmessage = amessage;
```

weist dem Zeiger **pmessage** lediglich die Adresse der Zeichenkette "**now is the time**" zu



- wird eine Zeichenkette als aktueller Parameter an eine Funktion übergeben, erhält diese eine Kopie des Zeigers



■ Zeichenketten kopieren

```
/* 1. Version */
void strcpy(char s[], t[])
{
    int i=0;
    while ( (s[i] = t[i]) != '\0' )
        i++;
}

/* 2. Version */
void strcpy(char *s, *t)
{
    while ( (*s = *t) != '\0' )
        s++, t++;
}

/* 3. Version */
void strcpy(char *s, *t)
{
    while ( *s++ = *t++ )
        ;
}
```



Felder von Zeigern

- Auch von Zeigern können Felder gebildet werden

- Deklaration

```
int *pfeld[5];  
int i = 1  
int j;
```

- Zugriffe auf einen Zeiger des Feldes

```
pfeld[3] = &i; ②
```

①

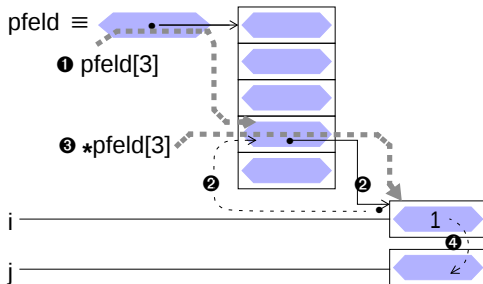
- Zugriffe auf das Objekt, auf das ein Zeiger des Feldes verweist

```
j = *pfeld[3]; ④
```

①

③

④

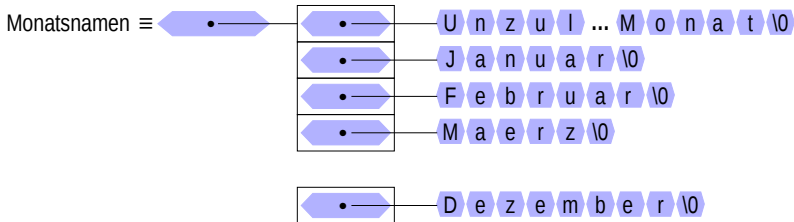


Felder von Zeigern (2)

- Beispiel: Definition und Initialisierung eines Zeigerfeldes:

```
char *month_name(int n)
{
    static char *Monatsnamen[] = {
        "Unzulaessiger Monat",
        "Januar",
        ...
        "Dezember"
    };

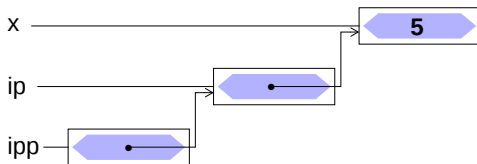
    return ( (n<0 || n>12) ?
             Monatsnamen[0] : Monatsnamen[n] );
}
```



Zeiger auf Zeiger

- ein Zeiger kann auf eine Variable verweisen, die ihrerseits ein Zeiger ist

```
int x = 5;  
int *ip = &x;  
  
int **ipp = &ip;  
/* → **ipp = 5 */
```



- wird vor allem bei der Parameterübergabe an Funktionen benötigt, wenn ein Zeiger "call bei reference" übergeben werden muss (z. B. swap-Funktion für Zeiger)



Argumente aus der Kommandozeile

- beim Aufruf eines Kommandos können normalerweise Argumente übergeben werden
- der Zugriff auf diese Argumente wird der Funktion **main()** durch zwei Aufrufparameter ermöglicht:

```
int  
main (int argc, char *argv[])  
{  
    ...  
}
```

oder

```
int  
main (int argc, char **argv)  
{  
    ...  
}
```

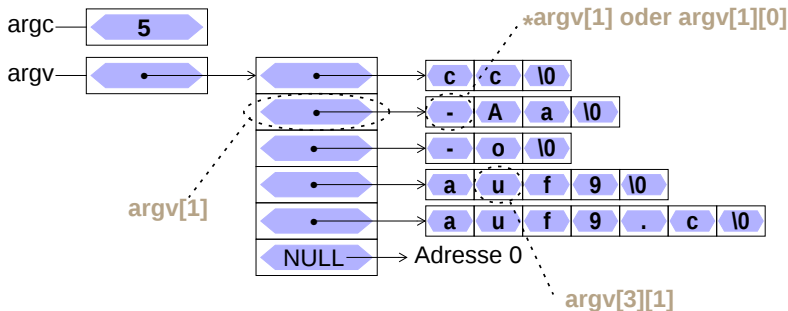
- der Parameter **argc** enthält die Anzahl der Argumente, mit denen das Programm aufgerufen wurde
- der Parameter **argv** ist ein Feld von Zeiger auf die einzelnen Argumente (Zeichenketten)
- der Kommandoname wird als erstes Argument übergeben (**argv[0]**)



Datenaufbau

Kommando: `cc -Aa -o auf9 auf9.c`

Datei cc.c:
...
`main(int argc, char *argv[]) {`
...



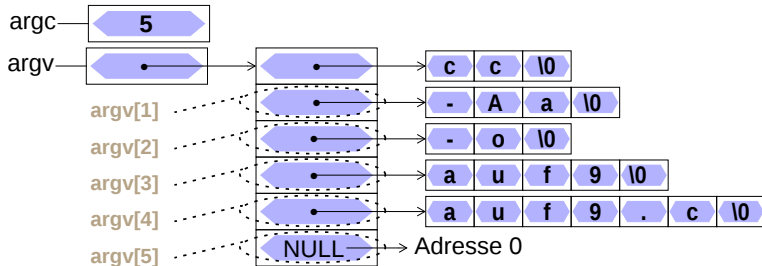
Zugriff

Beispiel: Ausgeben aller Argumente (1)

- das folgende Programmstück gibt alle Argumente der Kommandozeile aus (außer dem Kommandonamen)

```
int  
main (int argc, char *argv[])  
{  
    int i;  
    for ( i=1; i<argc; i++) {  
        printf("%s%c", argv[i], (i < argc-1) ? ' ':'\n' );  
    }  
    ...  
}
```

1. Version



Zugriff

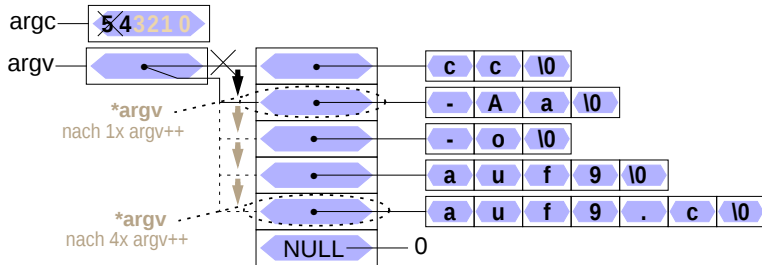
Beispiel: Ausgeben aller Argumente (2)

- das folgende Programmstück gibt alle Argumente der Kommandozeile aus (außer dem Kommandonamen)

```
int  
main (int argc, char **argv)  
{ while (--argc > 0) {  
    argv++;  
    printf("%s%c", *argv, (argc>1) ? ' ' : '\n' );  
}  
...
```

linksseitiger Operator:
erst dekrementieren,
dann while-Bedingung prüfen
→ Schleife läuft für argc=4,3,2,1

2. Version



Verbund-Datentypen / Strukturen (structs)

- Zusammenfassen mehrerer Daten zu einer Einheit
- Strukturdeklaration

```
struct person {  
    char name[20];  
    int alter;  
};
```

- Definition einer Variablen vom Typ der Struktur

```
struct person p1;
```

- Zugriff auf ein Element der Struktur

```
p1.alter = 20;
```



Zeiger auf Strukturen

- Konzept analog zu "Zeiger auf Variablen"
 - Adresse einer Struktur mit &-Operator zu bestimmen

- Beispiele

```
struct person stud1;  
struct person *pstud;  
pstud = &stud1;           /* ⇒ pstud → stud1 */
```

- Besondere Bedeutung zum Aufbau verketteter Strukturen



Zeiger auf Strukturen (2)

- Zugriff auf Strukturkomponenten über einen Zeiger

- Bekannte Vorgehensweise

- *-Operator liefert die Struktur
- .-Operator zum Zugriff auf Komponente
- Operatorenvorrang beachten



```
(*pstud).alter = 21;
```

nicht so gut leserlich!

- Syntaktische Verschönerung



->-Operator

```
pstud->alter = 21;
```



Rekursive Strukturen

- Strukturen in Strukturen sind erlaubt — aber
 - die Größe einer Struktur muss vom Compiler ausgerechnet werden können
 - Problem: eine Struktur enthält sich selbst
 - die Größe eines Zeigers ist bekannt (meist 4 Byte)
 - eine Struktur kann einen Zeiger auf eine gleichartige Struktur enthalten

```
struct liste {  
    struct student stud;  
    struct liste *rest;  
};
```

➡ Programmieren rekursiver Datenstrukturen



Überblick: Teil D Betriebssystemabstraktionen

15 Programmausführung, Nebenläufigkeit

16 Ergänzungen zur Einführung in C

17 Betriebssysteme I

18 Dateisysteme

19 Prozesse I

20 Speicherorganisation

21 Prozesse II

22 Betriebssysteme II



Was sind Betriebssysteme?

■ DIN 44300

- „...die Programme eines digitalen Rechensystems, die zusammen mit den Eigenschaften der Rechanlage die **Basis der möglichen Betriebsarten** des digitalen Rechensystems bilden und die insbesondere die **Abwicklung von Programmen steuern und überwachen**.“

■ Andy Tanenbaum

- „...eine Software-Schicht ..., die alle Teile des Systems verwaltet und dem Benutzer eine Schnittstelle oder eine *virtuelle Maschine* anbietet, die einfacher zu verstehen und zu programmieren ist [als die nackte Hardware].“

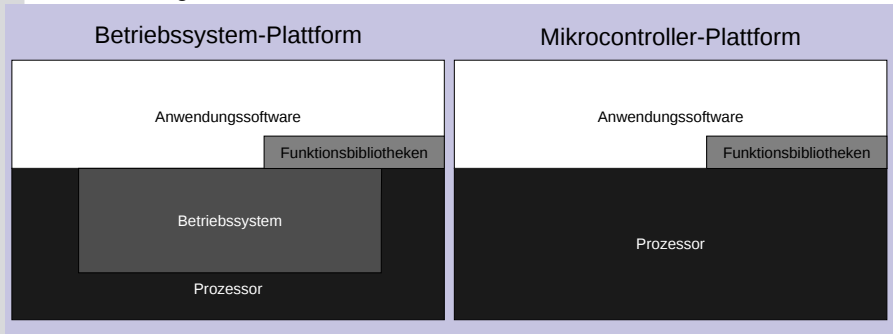
★ Zusammenfassung:

- Software zur Verwaltung und Virtualisierung der Hardwarekomponenten (Betriebsmittel)
- Programm zur Steuerung und Überwachung anderer Programme



Betriebssystem-Plattform vs. Mikrocontroller

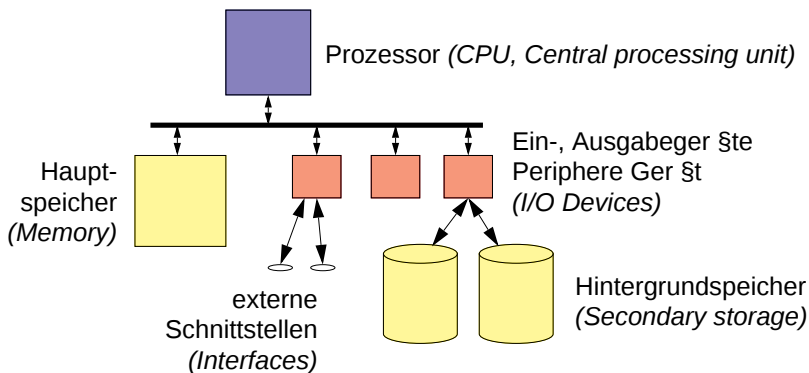
- Entscheidende Unterschiede:
 - Betriebssystem bietet zusätzliche Softwareinfrastruktur für die Ausführung von Anwendungen



- Software-Abstraktionen (Prozesse, Dateien, Sockets, Geräte, ...)
- Schutzkonzepte
- Verwaltungsmechanismen



Verwaltung von Betriebsmitteln



Verwaltung von Betriebsmitteln (2)

- Resultierende Aufgaben
 - Multiplexen von Betriebsmitteln für mehrere Benutzer bzw. Anwendungen
 - Schaffung von Schutzumgebungen
 - Bereitstellen von Abstraktionen zur besseren Handhabbarkeit der Betriebsmittel

- Ermöglichen einer koordinierten gemeinsamen Nutzung von Betriebsmitteln, klassifizierbar in
 - aktive, zeitlich aufteilbare (Prozessor)
 - passive, nur exklusiv nutzbare (periphere Geräte, z.B. Drucker u.Ä.)
 - passive, räumlich aufteilbare (Speicher, Plattenspeicher u.Ä.)

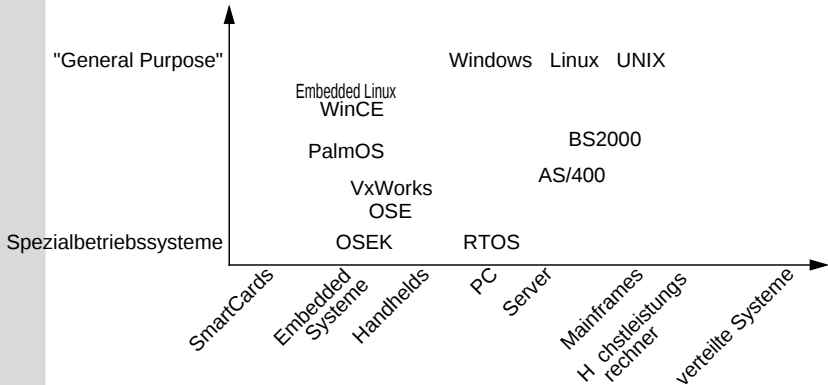
- Unterstützung bei der Fehlererholung



Klassifikation von Betriebssystemen

■ Unterschiedliche Klassifikationskriterien

- Zielplattform
- Einsatzzweck, Funktionalität



Klassifikation von Betriebssystemen (2)

- Wenigen "General Purpose"- und Mainframe/Höchstleistungsrechner-Betriebssystemen steht eine Vielzahl kleiner und kleinster Spezialbetriebssysteme gegenüber:

C51, C166, C251, CMX RTOS, C-Smart/Raven, eCos, eRTOS, Embos, Ercos, Euros Plus, Hi Ross, Hynet-OS, LynxOS, MicroX/OS-II, Nucleus, OS-9, OSE, OSEK Flex, OSEK Turbo, OSEK Plus, OSEKtime, Precise/MQX, Precise/RTCS, proOSEK, pSOS, PXROS, QNX, Realos, RTMOSxx, Real Time Architect, ThreadX, RTA, RTX51, RTX251, RTX166, RTXC, Softune, SSXS RTOS, VRTX, VxWorks, ...

- ➡ Einsatzbereich: Eingebettete Systeme, häufig Echtzeit-Betriebssysteme, über 50% proprietäre (in-house) Lösungen
- Alternative Klassifikation: nach Architektur



- Umfang zehntausende bis mehrere Millionen Befehlszeilen
 - Strukturierung hilfreich
- Verschiedene Strukturkonzepte
 - monolithische Systeme
 - geschichtete Systeme
 - Minimalkerne
 - Laufzeitbibliotheken (minimal, vor allem im Embedded-Bereich)
- Unterschiedliche Schutzkonzepte
 - kein Schutz
 - Schutz des Betriebssystems
 - Schutz von Betriebssystem und Anwendungen untereinander
 - feingranularer Schutz auch innerhalb von Anwendungen



Betriebssystemkomponenten

■ Speicherverwaltung

- Wer darf wann welche Information wohin im Speicher ablegen?

■ Prozessverwaltung

- Wann darf welche Aufgabe bearbeitet werden?

■ Dateisystem

- Speicherung und Schutz von Langzeitdaten

■ Interprozesskommunikation

- Kommunikation zwischen Anwendungsausführungen bzw. Teilen einer parallel ablaufenden Anwendung

■ Ein/Ausgabe

- Kommunikation mit der "Außenwelt" (Benutzer/Rechner)

