Programme, deren Zusammenwirken ‚wohldefinierte' Beziehungen der Rechner-Ressourcen* untereinander und zum Anwender (bzw.: Anwendung)** herstellen, bilden das Betriebs**system**.

*Komponenten; **Umgebung

Ein Betriebssystem erfüllt 2 Aufgaben (außen / innen):

- Übernahme der (schwierigen) Programmierung einzelner Hw-Komponenten und deren Zusammenspiels. Für den Anwender verhält sich das Betriebssystem wie eine Virtuelle Maschine.
  Wichtig: Einf. von Abstraktionsstufen / **Information Hiding**
- Bei mehrfacher gleichzeitiger Nutzung (Programme, Menschen): Regelung des Zugriffs auf Hw-Komponenten, Daten, Rechenzeit: Ressourcen-Verwaltung.

Neuer Übungsschwerpunkt gegenüber bisheriger Anwendungsprogrammierung

- Bestrebungen zur Hw-Abstraktion seit Z3 (K.Zuse, 1941)
- 1960er: Einf.v.Dialogbetrieb (vs. Stapelbetrieb / Operator)
  - Mitte der 60er: K.Thompson, D.Ritchie (Bell Labs, AT&T) und MIT-Forscher entwickeln MULTICS: Multiuser-BS f. Mainframe GE645
  - Kein Einsatz (tech.Mängel) ⇨ Weiterentwicklung durch Thompson („Space Travel“) ⇨ Verballhornung durch B.Kernighan: „UNICS“
  - 1969: Betriebssystem UNICS, ab 1970: Unix (in Assembler)

    Gleichzeitig (Thompson, Ritchie): A (BCPL-basiert) ⇨ B ⇨ C.
  - 1973: Unix erstes BS größtenteils in einer Hochsprache (kaum 1000 Zeilen Maschinencode) ⇨ Portierbarkeit!

    Keine Vermarktung durch AT&T wg. US-Kartell-Bestimmungen
  - Ab ca. 1975: Abgabe zum Selbstkostenpreis an Univers.: Univ. of California ⇨Berkeley Software Distribution (BSD) ⇨Erweiterungen
  - 1984: IEEE/POSIX⇨US-Standardisierungsvorgabe; 1988: ANSI-C
  - 1980-1990: Xenix (Microsoft, ab Mitte 80er: Santa Cruz Operation)
  - 1993: Windows NT; 1991: Linux 0.02 (Linus Torvalds, FIN)

Bemerkungen:

- **Information hiding**: Entwurfsprinzip, bei dem Module (=in sich geschlossene Software-Komponenten) als Black Boxes aufgebaut werden und somit ihren inneren Aufbau den Anwender/inne/n gegenüber verbergen. Dies erfolgt aus Gründen der Sicherheit (Manipulation, Copyright) oder zur Wahrung der Flexibilität (Änderungen unter Beibehaltung der Schnittstellen).
- Mit Win2000 erreichte Windows den Umfang v. 30 Millionen Codezeilen – u. d. US-Fachblatt Smart Reseller zufolge mit 63.000 potenziell bekannten Bugs.
- **ANSI**: American National Standards Institute (US-Normungsinstitution)
- **ASCII**: American Standard Code for Information Interchange (7-Bit-Code, 1968)
- **AT&T**: American Telephone & Telegraph Company (gegr. 1885)
- **BCPL**: Basic Combined Programming Language, entwickelt in den 60ern an den Univ. von London u. Cambridge (GB)
- **Lisa**: Local Integrated Software Architecture
- **MIT**: Massachusetts Institute of Technology, Boston
- **MS-DOS**: Microsoft Disk Operation System
- **MULTICS**: MULTiplexed Information and Computing System
- **UNICS**: UNiplexed Information and Computing System. Der Scherz bestand darin, daß, während Multics mehrere Alternativen zur Erlangung seines Ziels untersuchte, Unics sich nur auf eine einfache konzentrierte.
- **Xerox PARC**: Palo Alto Research Center der Xerox Corporation

Betriebssysteme

- Heute: ca. ein Dutzend Betriebssysteme; Marktführer: MS
- Gründe für d. PC-Erfolg (Debüt: IBM 1981/ mit MS-DOS)
  - Mächtiger Anbieter (IBM):
    Gewähr für Fortbestand & Anbindung an Großrechner
  - Modulares Bau-Konzept:
    Nach Bedarf aufrüstbar & Günstige Nachbauten Dritter
  - Orientierung am Massenmarkt („Schneeball-Effekt“):
    Erschwinglicher Preis & Kurze Einarbeitung
- Paradigmenwechsel im Anspruch an Rechner:
  ≤80er: Erzeugung v. Ergebnissen; Darstellung auf Drucker
  heute: Ständiger Dialog; Ergebnisse in div. Formen/Medien
- Wichtig für die Realisierung: Forschung am Xerox PARC
  Mitte 70er: Konzept für graf. Benutzungsschnittstelle (GUI)
  Apple: 1983 „Lisa“, '84 Macintosh; MS:1985 Windows 1.01

Grafische Benutzungsschnittstellen

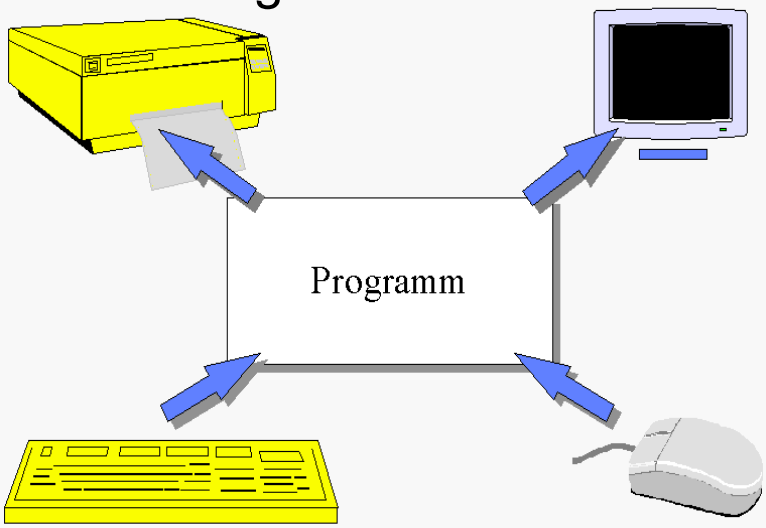
- Ziel der GUI: Verlagerung des Schreibtisches in den PC
  „Desktop“: Rechner auf dem Tisch – eher: Tisch-Oberfläche!
  Vorstellung: Papier-Stapel („Batch“) vs. Lose Blätter (Fenster)
- Einige Forderungen an GUI und ihre Auswirkungen
  - Intuitive Handhabung ⇨ Minimierung d. Voraussetzungen
    (z.B.: Brief-/Mail-Schreiben nach Klick statt Befehl-Eingabe)
    ➥ Design / Ergonomie
  - Quasi-parallele Bearbeitung ⇨ Präemptives Multitasking
    (z.B.: Erstellung v. Präsentation während Internet-Suche)
  - Arbeits-Abläufe einer Person ⇨ Prozeß-Kommunikation
    (z.B.: Datei-Löschen i. allen betroffenen Fenstern anzeigen)
    ➥ Plattform / Betriebssystem
  - Einheitliches „Look & Feel“ ⇨ Kapselung d. Ein-/Ausgabe
    (z.B.: „Drag&Drop“ programm-übergreifend realisiert)
    ➥ Applikationen / Bibliotheken

Grafische Benutzungsschnittstellen

⇨ Steigerung d. Hw-Abstraktion durch zusätzliche Sw

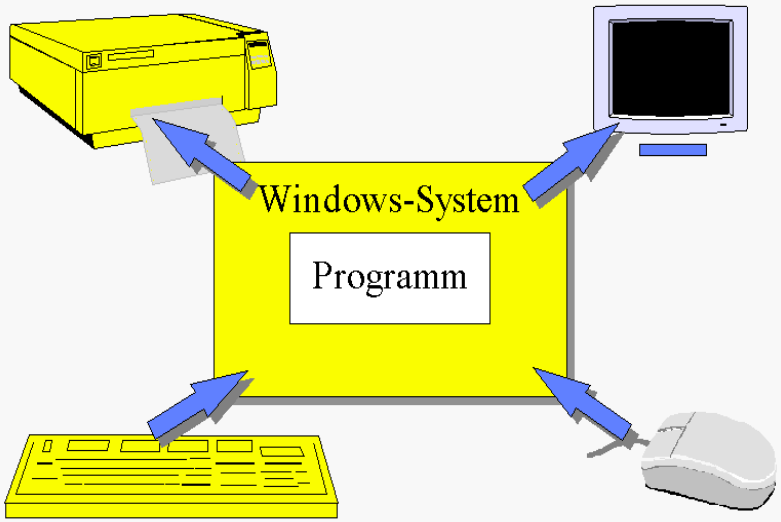
Anwendungsprogrammierung unter **MS-DOS**:

"Standard-I/O" u.standardisierte Rechenzeit-Zuweisung; abweichende Handhabung durch Programmierer/in



Windows-Programmierung:

Kommunikation mit Peripherie ins System integriert: Geräte werden "wie unter Windows" angesprochen.



Bilder: W. Doberenz, Th. Kowalski: "Programmieren lernen in Visual Basic 5", Hanser 1997

Zur Verdeutlichung: Word for DOS auf ≥20 Disketten ausgeliefert; darunter: 1-2 für das Programm selbst, Rest für diverse Treiber

- Dialog-Komponenten immer komplexer / differenzierter
  (z.B.: gezielte Warn-Meldungen mit Fallunterscheidungen)
- Dialog (quasi-)unabhängig vom Programm
  (z.B.: Menüleisten mit: „Datei - Bearbeiten - Ansicht -...“)

⇨ Loslösung der eigentlichen Applikation von Ein-/Ausgabe

⇨ Abgabe der Ablaufkontrolle an GUI-Umgebung/-Plattform

Konsequenzen für die Applikations-Entwicklung:

- für die Ergonomie: sehr groß
- für das DV-Ergebnis: keine
- für die Sw-Konzeption: gering
- für die Codierung: deutlich (v.a. für kleine Sw-Projekte)

Zusätzlicher Codierungsaufwand durch Berücksichtigung

- der Plattform-Aufrufe
  (z.B.: Ausgabe im Fenster) - aber auch
- der plattform-eigenen Erfordernisse (Plattform-‘Eigenleben‘)
  (z.B.: Änderung der Fenstergröße, Löschung des Fensters)

Beispiel / Übung:

Beispiel-Projekt:

Sw-Umgebung KSP (‚KataStrophen-Prog') soll I/O übernehmen.

Anwendungsprogramm:

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

int main (void)
{ int  num=0, ch=' ';
  while ((ch = _getch()) != 27)
  { num++;  printf ("%d\n", num);
  }
  return (num);
}
```

(App0th.exe)

Dringend empfohlen: Trennung & Modularisierung ⇨

Beispiel / Übung:

`main()` entbehrlich halten:

```
int data=0;
/* Zu Testzwecken: */
int main (void)
{
  printf("\n\r A.Christidis:");
  printf("\n\r AnschlgZaehlg");
  printf("\n\r Ende: <Esc>");
  printf("\n\r(bel. Taste)");
  _getch();
  /*Besser-z.B.: App4Win.h mit
  #define CLS "cls" */
  system("cls"); /*system(CLS);*/

  example ();
}
```

(App1st.exe)

```
/*Beispielhafte Anwendung:*/
int example (void)
{ int ch=' ';

  while ((ch=_getch()) != 27)
  { data = calc (data);
    printf ("%5d\n", data);
  }
  return (data);
}
```

I/O

```
/*Fachlich relevant:*/
int calc (int num)
{ return (++num);
}
```

Algorithmus bleibt unberührt!

Neue I/O-Aufrufe:
```
int KSPgetch(void);
int KSPprintf(char *format, int data);
```

Beispiel / Übung:


```
int   data=0;      /*Daten ...*/
char *form="%5d"; /*... und ihr Ausgabe-Format*/
/* Beispielhafte Anwendung: */
int example (void)
{ int  ch=' ';
  KSPinit (redisplay);
  while ((ch = KSPgetch()) != 27)
  { data = calc (data);
    KSPprintf (form, data);
  } return (data);
}
```

Wichtig: Funktion als Maßnahme bei Löschung des Fensters:

```
/* Anpassung an Plattform-Vorgaben: */
int redisplay (void)              /* Callback-Funktion*/
{ int j1=MAX((data-KSPLINES),0);//#define KSPLINES 24
  while (j1 < data)
  { j1= calc (j1);  KSPprintf (form, j1);
  }  return (j1);
} /* Bekanntgabe bei der Initialisierung!! */
```

Callback

Zentraler Begriff bei Errichtung o. Nutzung v. Sw-Plattformen:

➡ **Callback** [-Funktion]: Funktion d. Anwendungsprogramms, die in vorgegebenen Situationen von der Sw-Umgebung aufgerufen wird (z.B. zum Auffrischen des Fensterinhalts)

Hintergrund:

- Multitasking ⇔ Teilen von Ressourcen mit anderen (I/O, Rechenzeit etc.)
- Koordination nur durch Sw-Umgebung / -Plattform möglich (Info über laufende Programme u. Ressourcen-Bedarf)

Callback

⇨ Maßnahmen nach Wiederzuweisung von Ressourcen sind nur durch die Plattform einzuleiten – aber:

➡ Code ist nur als Bestandteil der Applikation sinnvoll: nur dort ist Information über Wiederherstellungs-Maßnahmen vorhanden – z.B.:

Uhr: . . . . . Abfrage (ggf. vorzeitig)

Bild: . . . . . Neuladen

Live-TV: . . . . . (keine Maßnahme)

Wie kann die (Standard-) Applikation der (Standard-) Plattform mitteilen, welche Funktion aufzurufen ist?

- Einrichtung einer Plattform-Routine zur Aufnahme eines Zeigers auf eine C-Funktion

(„Callback“ = Rückruf)

Callback

Zeiger auf C-Funktionen:

Passend (Typ, Parameterliste) zu einer Funktion – z.B.:

```
int myfunc (void)
```

kann eine Zeiger-Variable **p2f** deklariert werden,...

```
int (*p2f)(void)    //bis C99: int (*p2f)();
```

Ohne Klammer: Variable **int *p2f**

Ohne Klammer: Return-Wert **int***

Ohne **void**: bel. Parameter-Liste

... die zur Laufzeit die physikalische (Einsprungs-)Adresse der Funktion zugewiesen bekommt:

```
p2f = myfunc;
```

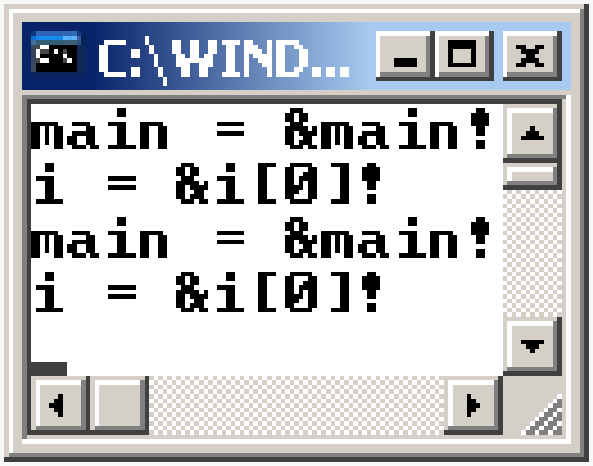
(vgl. Felder: Adreßoperator **& unnötig**)

Die Funktion kann nun über ihre Adresse aufgerufen werden:

```
(*p2f)();              //seit C99 auch: p2f();
```

Callback

Beispiel / Experiment:




```
#include <conio.h>   //wg. _getch()
#include <stdio.h>   //wg. printf()

int main(void)
{ static int i[1]={0};
  /*Funktionsname = Funktionsadresse...*/
  if (main == &main) printf ("main = &main!\n");

  /*...wie Feldname = Feldadresse:*/
  if (i == &i[0]) printf ("i = &i[0]!\n");

  /*Wiedereintritt ohne Endlos-Schleife:*/
  if (i[0]++ < 1) (*main)();      /*wie: main();*/
  _getch();
  return 0;
}
```

(Callback0\Exc\Test.exe)

Callback

Zeiger auf C/C++-Funktionen (2):

```
#include <stdio.h>  /*printf(), getchar()*/
```

Plattform

```
void (*funcPointer)(int)=NULL;
/* Callback-Registrierung: */
void init (void (*callback)(int))
{ printf ("In init()!\n");
  callback (0);      /*  untypisch, aber OK  */
  funcPointer = callback;
 (*funcPointer)(1); /*i.d.R. in anderen Fktn*/
  return;
}
```

Applikation

```
/* Callback: */
void myfunc(int param)
{ printf ("In myfunc() mit %d!\n", param);
  getchar();  return;
}
int main (void)
{ init (myfunc);
  return 0 ;
}
```

(Callback1\Exc\Test.exe)

Ereignisse

Verstärkter Einsatz von Callbacks in modernen Plattformen und Umgebungen führt zu weiterem Paradigmenwechsel:

Klassisch:

Programm fordert vom BS Ressourcen u. Aktionen an

(prozedurorientierte Arbeitsweise - z.B.: Single Task)

Nunmehr meist:

Betriebssystem gibt Programm Bescheid, wann es und mit welcher Funktion „an der Reihe ist”

(ereignisgesteuerte Arbeitsweise)

Beispiel:

I/O-Kontrolle ermöglicht bei gleichem **`example`** und **`calc`** veränderte Ausgabe: `(App2nd.exe)`

Ereignisse

- Definition: Als **Ereignis** (*engl. event*) bezeichnen wir jedes Vorkommnis *(occurence)*, das eine nicht-sequentielle Bearbeitung eines Programms bewirkt.

  Ein Vorkommnis kann eine Zustandsänderung (Meßwert) oder die Erfüllung einer Bedingung (`x == y`) sein.

  Ein dazugehöriges Ereignis kann die Abarbeitung einer `if`-Abfrage im Programm sein.

- **Synchrone** Ereignisse treten zu vorhersagbaren Zeitpunkten, **asynchrone** zu nicht-vorhersagbaren ein.

  Synchrone Ereignisse sind z.B. `if`-Abfragen im prozedur-orientierten Programm: sie treten vorhersagbar ein (z.B.: ab Programmbeginn).

  Asynchrone Ereignisse sind z.B. Tasten- u. Maus-Aktionen (Zeit oft nicht-vorhersagbar: „OK“ vs. „Abbrechen“).

d.h.: Programm-stelle!

Ereignisse

Ereignisgesteuerte Arbeitsweise bedient sich asynchroner Ereignisse.

Klassische Auslöser asynchroner Ereignisse:

- die Systemzeit (C-Anweisungen **`clock()`**, **`time()`**)

```
clock_t start;  start=clock();
// Rechenzeit seit Start [sec*CLOCKS_PER_SEC]
time_t  start;  time(&start);
// Wartezeit seit 01.01.1970 00:00:00 [sec]
```

- die Tastatur (Nicht-ANSI-Anweisung **`_kbhit()`**),
- die Maus (nicht über C ansprechbar)

und

- Kombinationen daraus.

Übung:

Aufgaben:

1. Konzipieren und codieren Sie die neue Sw-Umgebung KSP so, daß Sie **`int _getch(void)`** durch **`int KSPgetch(void)`**, **`int printf (const char* format, int data)`** durch **`int KSPprintf (char *format, int data)`** ersetzen können.

   Zur Initialisierung erstellen Sie
   `void KSPinit (int (*callback)(void))` .
2. Verändern Sie die Umgebung so, daß die Ausgabe durch Betätigung der Tasten ‚s' und ‚d' verschoben wird.
3. Verändern Sie die Umgebung so (bedingte Kompilierung), daß die Applikation nur 30 sec aktiv bleibt - bei eingeblendetem Countdown.
4. Beseitigen Sie alle globalen Variablen, die Sie evtl. verwendet haben.

`App3rd.exe`