

# Script generated by TTT

Title: Grundlagen\_Betriebssysteme (26.10.2015)

Date: Mon Oct 26 13:53:32 CET 2015

Duration: 81:31 min

Pages: 24

## Betriebssystem-Architektur

In der Praxis findet man einige verschiedene BS-Architekturkonzepte, wobei der monolithische Ansatz und zunehmend auch der Mikrokern-Ansatz am weitesten verbreitet sind.

- [Monolithischer Ansatz](#)
- [Mikrokern-Ansatz](#)

Beispiel: BS-Architekturen

- [Unix Betriebssystem](#)
- [Windows NT Betriebssystem](#)
- [Linux Betriebssystem](#)

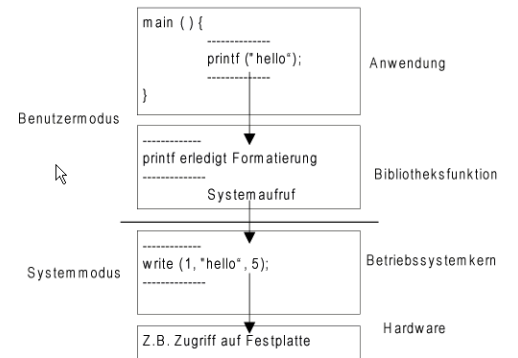
Systemaufrufe

[Virtuelle Maschine](#)

Generated by Targem.com

## Systemaufrufe

Aufruf von BS Funktionalität durch Anwenderprogramme mittels Systemaufrufe.  
in Benutzerprogrammen werden Systemaufrufe nicht direkt verwendet, sondern dies erfolgt über die Einbindung von Systembibliotheken, z.B. C-Library.



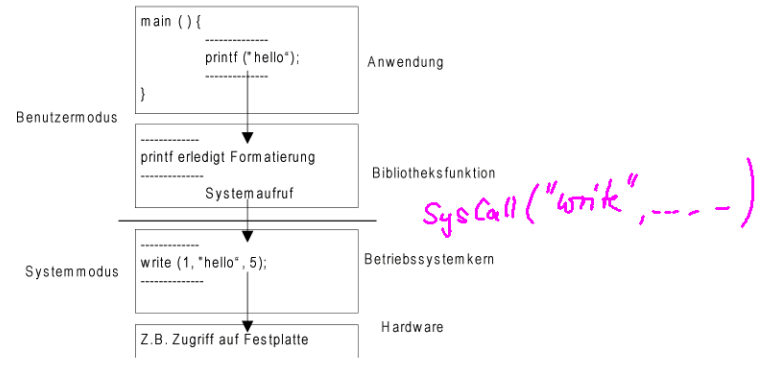
Systemaufruf führt zum Übergang vom Benutzermodus in den Systemmodus.

Beispiele von Systemaufrufen

- Prozessmanagement: fork, waitpid, exit.
- Dateimanagement: open, close, read, write.
- Verzeichnismangement: mkdir, rmdir, link, mount.
- Gerätemanagement: request/release device, get/set device attributes.

## Systemaufrufe

Systembibliotheken, z.B. C-Library:



Systemaufruf führt zum Übergang vom Benutzermodus in den Systemmodus.

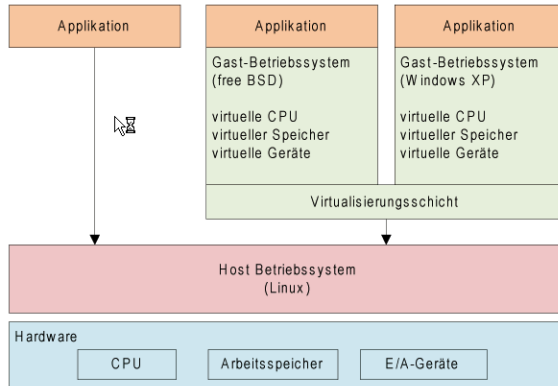
Beispiele von Systemaufrufen

- Prozessmanagement: fork, waitpid, exit.
- Dateimanagement: open, close, read, write.
- Verzeichnismangement: mkdir, rmdir, link, mount.
- Gerätemanagement: request/release device, get/set device attributes.
- Kommunikationsmanagement: send/receive messages, create/delete connection.



Virtualisierungskonzept erlaubt die gleichzeitige Bereitstellung mehrerer Betriebssysteme auf einem Rechner.  
 Isolierung der virtuellen Maschinen.  
 gemeinsame Nutzung von Dateien möglich.

Beispiel **VMware**.



Java Virtual Machine (JVM): abstrakter Computer zur Ausführung von Java Programmen; implementiert in Software.

*Generated by Targeteam*



**Maschinenschnittstelle**

Als Maschinenschnittstelle bezeichnet man die Gesamtheit aller Datenobjekte und Operationen der reinen Hardwarearchitektur (auch Programmierschnittstellen der Maschine).

Folge von Maschinenbefehlen ist auf dieser Ebene eine Folge von Binärzeichen.

**Assemblerschnittstelle**

Die Assemblerschnittstelle ist die eigentliche maschinennahe (konkrete) Programmierschnittstelle. Sie erlaubt, alle Befehle, Adressen und Datenobjekte der reinen Hardware darzustellen.

Verwendung von Namen für Adressen und Operationen.

**Assembler**

Ein Assembler ist ein Programm, das die Aufgabe hat,

1. Assemblerbefehle in Maschinencode zu transformieren,
2. symbolischen Namen Maschinenadressen zu zuweisen, sowie
3. ein Objektprogramm zu erzeugen.

*Generated by Targeteam*



In der Praxis findet man einige verschiedene BS-Architekturkonzepte, wobei der monolithische Ansatz und zunehmend auch der Mikrokern-Ansatz am weitesten verbreitet sind.

Monolithischer Ansatz

Mikrokern-Ansatz

**Beispiel: BS-Architekturen**

Unix Betriebssystem

Windows NT Betriebssystem

Linux Betriebssystem

Systemaufrufe

Virtuelle Maschine

*Generated by Targeteam*



Nutzung von Assemblerprogrammen, aber auch C-Programmen zur Realisierung hardwarenaher Programme für ein Rechensystem.

Definitionen

Programmaufbereitung

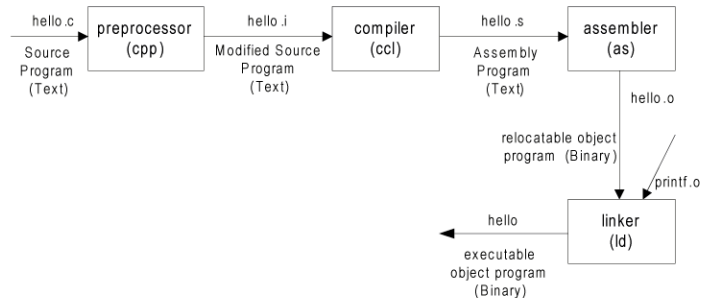
Binder und Lader

*Generated by Targeteam*



Aufruf:

```
unix> gcc -o hello hello.c
```

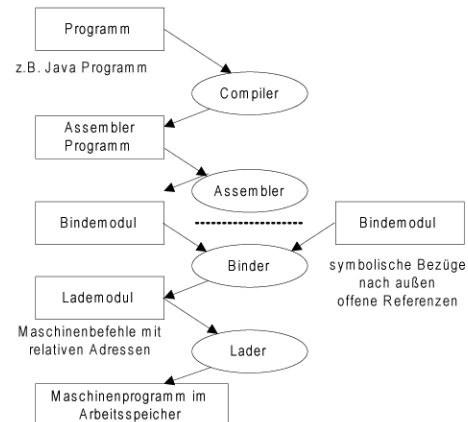


Transformation in 4 Phasen

- Preprocessor: Aufbereitung.
- Compiler: Übersetzer von C nach Assembler.
- Assembler: Übersetzer von Assembler nach Maschinsprache; Generierung des Objektprogramms.
- Linker (Binder): Nachbearbeitung und Kombinationen verschiedener Module.

Generated by Targeteam

Programmieren auf der Assemblerschnittstelle; Verwendung von Assembler-Programmen

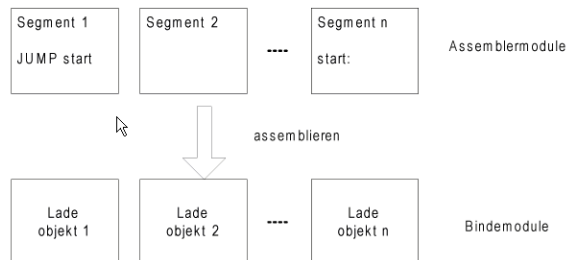


Beispiel Unix Compilersystem

Generated by Targeteam



Der Assembler erzeugt Code, der jeweils relativ zum Modul-Anfang adressiert.



Das Zusammenfügen der einzelnen Ladeobjekte zu einem ausführfähigen Programm ist die Aufgabe des Binders (Linker).

Externe Referenzen

In einem Modul M<sub>i</sub> können Referenzen auf Daten/Symbole auftreten, wobei die Daten/Symbole in einem anderen Modul M<sub>j</sub> definiert werden. Beispiel: Symbol **start** wird in Segment 1 verwendet, und erst in Segment n definiert.

Behandlung externer Referenzen

Generated by Targeteam

Für jede externe Referenz erzeugt der Assembler Informationen, die es dem Binder ermöglicht, aus den Einzelmodul ein ausführfähiges Gesamtprogramm zu erzeugen, d.h. die Referenzen aufzulösen.

- Assembler-Anweisungen für den Export von Symbolen.
- Assembler-Anweisungen für den Import von externen Symbolen.
- Assembler erzeugt ausgehend von diesen Steuerinformationen spezielle Einträge im Objekt-Programm.
  - Für **exportierte Symbole**: ein Eintrag (gekennzeichnet z.B. durch ein D (=define)), der den Namen und die relative Adresse des Symbols umfasst.
  - Für **importierte Symbole**: ein Eintrag (gekennzeichnet z.B. durch ein R (=refer)), der den Namen des importierten Symbols umfasst.
- Auftreten einer externen Referenz: Eintrag der Adresse 0 in erzeugtem Code und Generierung eines Modifikations-Eintrags:
  - Angabe, welches Auftreten der Referenz zu modifizieren ist und
  - Name des externen Symbols.

Generated by Targeteam



## Binder und Lader



Ein Assembler-Programm kann aus mehreren logischen Einheiten bestehen, die vom Assembler als einzelne, unabhängige Einheiten (Bindemoduln) transformiert werden.

### Binder

Der **Binder** (engl. linker) hat die Aufgabe, aus einer Menge von einzelnen Bindemoduln ein ausführfähiges Ladeprogramm zu erzeugen, indem die noch offenen externen Referenzen aufgelöst werden.

#### Binde-Module

### Lader

Ein **Lader** (engl. loader) ist ein Systemprogramm, das die Aufgabe hat, Objektprogramme in den Speicher zu laden und deren Ausführung anzustoßen.

#### Eigenschaften

#### Datenstrukturen eines Binde/Laders

#### Dynamisches Binden

Generated by Targeteam



## Eigenschaften



In einem System ist i.d.R. nur ein Lader vorhanden, so dass Programme unterschiedlicher Quellsprachen in ein einheitliches Objektprogramm-Format transformiert werden müssen.

Viele Lader beinhalten gleichzeitig Binde-Funktion. **Binde/Lader** sind heutzutage typische Komponenten in Rechensystemen. Binde/Lader sind Bestandteil der Dienste, die ein Betriebssystem anbietet.

Binde/Lader: Programmmodule werden zur Ladezeit gebunden.

**Lauf 1** : Zuweisung von Adressen zu externen Referenzen (Auflösen von Referenzen).

**Lauf 2** : Binden, Verschieben, Laden.

#### Absoluter Lader

#### Relativer Lader

Laden verschiebbarer Objekt-Programme, wobei die Information, welche Adressen neu zu berechnen sind, vom Assembler zur Verfügung gestellt werden, z.B. durch Modifikations-Einträge im Objekt-Programm.

Generated by Targeteam



## Absoluter Lader



Aufgaben eines absoluten Laders; ein Lauf genügt.

Prüfe, ob der für das Programm vorgesehene Speicher groß genug ist.

Die Text-Einträge im Objekt-Programm werden gelesen und der Code wird an die dort angegebenen, absoluten Adressen des Speichers geladen.

Beim Lesen des END-Eintrags springt der Lader zur angegebenen Start-Adresse des Programms, um die Programmausführung zu starten.

Generated by Targeteam



## Eigenschaften



In einem System ist i.d.R. nur ein Lader vorhanden, so dass Programme unterschiedlicher Quellsprachen in ein einheitliches Objektprogramm-Format transformiert werden müssen.

Viele Lader beinhalten gleichzeitig Binde-Funktion. **Binde/Lader** sind heutzutage typische Komponenten in Rechensystemen. Binde/Lader sind Bestandteil der Dienste, die ein Betriebssystem anbietet.

Binde/Lader: Programmmodule werden zur Ladezeit gebunden.

**Lauf 1** : Zuweisung von Adressen zu externen Referenzen (Auflösen von Referenzen).

**Lauf 2** : Binden, Verschieben, Laden.

#### Absoluter Lader

#### Relativer Lader

Laden verschiebbarer Objekt-Programme, wobei die Information, welche Adressen neu zu berechnen sind, vom Assembler zur Verfügung gestellt werden, z.B. durch Modifikations-Einträge im Objekt-Programm.

Generated by Targeteam



Tabelle ESTAB ("external symbol table") zur Auflösung externer Referenzen; Tabelleneintrag = [Symbol, Adresse].

#### Hilfsvariable

**PADR** : Startadresse im Speicher, wohin das gebundene Programm geladen werden soll.

**CSADR** : Startadresse des jeweils bearbeiteten Moduls; dieser Wert wird zu den Relativ-Adressen des jeweiligen Moduls hinzu addiert.

#### [Algorithmus zum Lauf 1](#)

#### [Algorithmus zum Lauf 2](#)

*Generated by Targeteam*



Festlegen der Startadresse PADR des zu ladenden Programms (also wohin es in den Speicher geladen werden soll). Jedes Modul wie folgt bearbeiten:

Header-Eintrag lesen und Eintrag in Symboltabelle ESTAB:

[Name des Moduls, Startadr. (=CSADR) des Moduls].

Beim ersten Modul gilt: CSADR=PADR.

Lesen von Export-Einträgen (Symbol-Definitionen) im Objekt-Programm; alle auftretenden Symbole in ESTAB eintragen, wobei gilt:

[symbolischer-Name, Adresse = Relativadr + CSADR].

Lesen des END-Eintrags: CSADR = CSADR\_alt + Länge des Segments (steht als Info im Header); Bearbeiten des nächsten Moduls mit der neuen Anfangsadresse CSADR.

*Generated by Targeteam*



Nach Lauf 1 enthält ESTAB alle externen Symbole, die in Modulen definiert wurden zusammen mit deren Adresse.

Sukzessives Lesen der Text-Einträge aus Objektprogramm; Abspeichern des Codes an "Startadresse des Segments + Relativadresse" im Eintrag.

Wird ein Modifikations-Eintrag gelesen, so wird das extern referenzierte Symbol bzw. dessen Adresse, in ESTAB nachgeschlagen.

Ist das letzte Modul bearbeitet und dessen END-Eintrag gelesen, so wird zum dort angegebenen Beginn des Programms gesprungen und die Kontrolle zur Ausführung des Programms wird an das Programm abgegeben.

*Generated by Targeteam*



Binden von Unterprogrammen erst zur Laufzeit, d.h. erst wenn sie das erste Mal aufgerufen werden. Als Vorteile ergeben sich folgende:

#### Nach Bedarf laden

Unterprogramme werden also nur dann, wenn sie tatsächlich gebraucht werden, zum in Ausführung befindlichen Programm hinzu gebunden.

Beispiel Windows: Routinen in Dynamic Link Libraries (DLL) zusammengefasst; erst wenn sie benötigt wird, Laden der gesamten DLL.

#### Code-Sharing

Dynamisches Binden wird oft verwendet, wenn mehrere ausführfähige Programme eine einzige Kopie eines Unterprogramms oder einer Bibliothek gemeinsam nutzen sollen.

*Generated by Targeteam*





Ein Assembler-Programm kann aus mehreren logischen Einheiten bestehen, die vom Assembler als einzelne, unabhängige Einheiten (Bindemoduln) transformiert werden.

### Binder

Der **Binder** (engl. linker) hat die Aufgabe, aus einer Menge von einzelnen Bindemoduln ein ausführfähiges Ladeprogramm zu erzeugen, indem die noch offenen externen Referenzen aufgelöst werden.

#### Binde-Module

### Lader

Ein **Lader** (engl. loader) ist ein Systemprogramm, das die Aufgabe hat, Objektprogramme in den Speicher zu laden und deren Ausführung anzustoßen.

#### Eigenschaften

#### Datenstrukturen eines Binde/Laders

#### Dynamisches Binden

Generated by Targteam



In einem allgemeinen Rechensystem finden eine Vielzahl paralleler Abläufe statt. Dieser Abschnitt beschäftigt sich mit den Eigenschaften paralleler Systeme.

### Fragestellungen

#### Grundlagen

#### Modellierung paralleler Systeme

#### Thread-Konzept

#### Synchronisation

#### Verklemmungen

Generated by Targteam



Sequentielle Aspekte von systemnaher Programmierung:

Programm als Repräsentation eines Algorithmus;

Programm als **sequentielle** Folge von Aktionen/Anweisungen (alle Schritte nacheinander);

**determinierte** Programme: unter gleichen Bedingungen und Eingaben produziert das Programm stets das gleiche Ergebnis;

**deterministische** Abläufe: eindeutig vorbestimmter Ablauf, keine willkürliche Auswahl von alternativen Schritten.

Übergang von sequentiellen Systemen hin zu parallelen Systemen/Programmen. Dabei werden die folgenden Aspekte näher betrachtet.

gleichzeitige Aktivität von Komponenten, die möglicherweise miteinander kommunizieren oder aufeinander einwirken.

### **Beispiele für Parallelität**

HW-Komponenten eines Rechensystems, z.B. Prozessor und E/A-Controller.

SW-Komponenten eines Rechensystems, z.B. parallel ablaufende Programme; u.U. Zugriff auf gemeinsame Ressourcen, z.B. Daten im Arbeitsspeicher.

SW-Komponenten in einem verteilten System, d.h. Rechensysteme, die über ein Rechnernetz miteinander verbunden sind.

#### Aspekte des Abschnitts

Generated by Targteam



Sequentielle Aspekte von systemnaher Programmierung:

Programm als Repräsentation eines Algorithmus;

Programm als **sequentielle** Folge von Aktionen/Anweisungen (alle Schritte nacheinander);

**determinierte** Programme: unter gleichen Bedingungen und Eingaben produziert das Programm stets das gleiche Ergebnis;

**deterministische** Abläufe: eindeutig vorbestimmter Ablauf, keine willkürliche Auswahl von alternativen Schritten.

Übergang von sequentiellen Systemen hin zu parallelen Systemen/Programmen. Dabei werden die folgenden Aspekte näher betrachtet.

gleichzeitige Aktivität von Komponenten, die möglicherweise miteinander kommunizieren oder aufeinander einwirken.

### **Beispiele für Parallelität**

HW-Komponenten eines Rechensystems, z.B. Prozessor und E/A-Controller.

SW-Komponenten eines Rechensystems, z.B. parallel ablaufende Programme; u.U. Zugriff auf gemeinsame Ressourcen, z.B. Daten im Arbeitsspeicher.

SW-Komponenten in einem verteilten System, d.h. Rechensysteme, die über ein Rechnernetz miteinander verbunden sind.

#### Aspekte des Abschnitts

Generated by Targteam



Modellierungstechniken zur Analyse und Beschreibung der Eigenschaften paralleler Systeme (Spuren, Petrinetze).  
Threads als Mittel zur Realisierung von Parallelität innerhalb eines Prozesses.  
Synchronisation in parallelen Systemen.  
Mechanismen zur Behandlung von Verklemmungen.

*Generated by Targemam*

