Script **generated by TTT**

Title: Grundlagen_Betriebssysteme (26.10.2012)

Date: Fri Oct 26 08:28:40 CEST 2012

Duration: 88:20 min

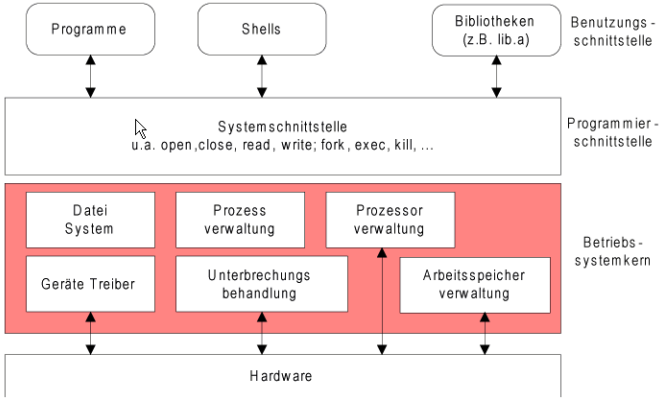
Pages: 28

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.

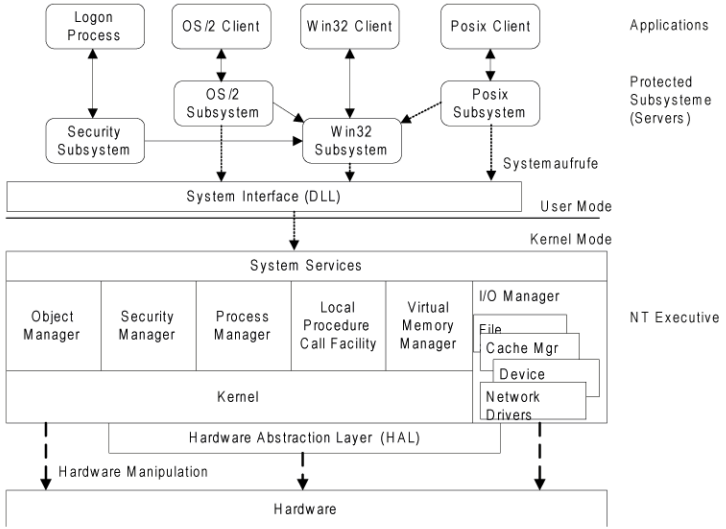
Generated by Targeteam

Die nachfolgende Abbildung skizziert die wesentlichen Komponenten des Unix Betriebssystems. Der Unix-BS-Kern enthält die Datei-, Prozess- und Prozessorverwaltung, die Speicherverwaltung und die Geräte-Treiber.



Generated by Targeteam

Mit Hilfe von ***HAL*** wird versucht, die meisten Maschinenabhängigkeiten zu verbergen. HAL präsentiert dem restlichen BS abstrakte Hardwaregeräte (z.B. Systembus, Arbeitsspeicher etc).



Generated by Targeteam

Linux Betriebssystem

Linux begann Anfang der 1990er Jahre als eine Unix Variante für den IBM PC;

erste Version durch Linus Torvalds (1991).

Linux ist frei und Quellcode ist verfügbar.

kollaborative Weiterentwicklung durch Open Source Community.

kein Mikrokern-Ansatz, jedoch modulare Struktur.

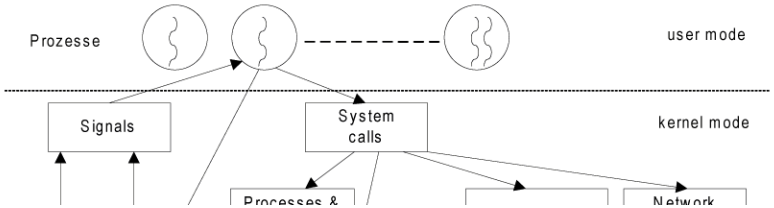
dynamic linking.

Module sind hierarchisch organisiert.

jeder Modul ist durch 2 Datenstrukturen beschrieben.

Modulbeschreibung, u.a Modulname, Größe, Zahl der exportierten Symbole und referenzierte Module.

Symbol-Tabelle.

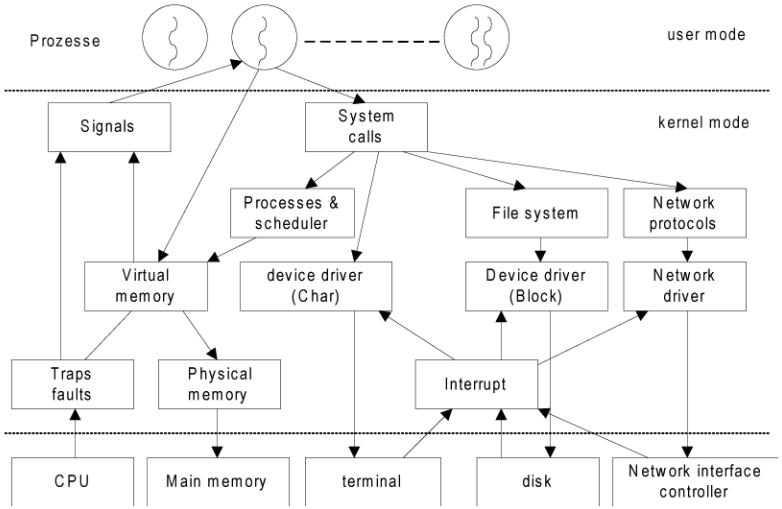


Linux Betriebssystem

Modulbeschreibung, u.a Modulname, Größe, Zahl der exportierten Symbole und referenzierte Module.

Symbol-Tabelle.

Liste der exp. Symbole



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

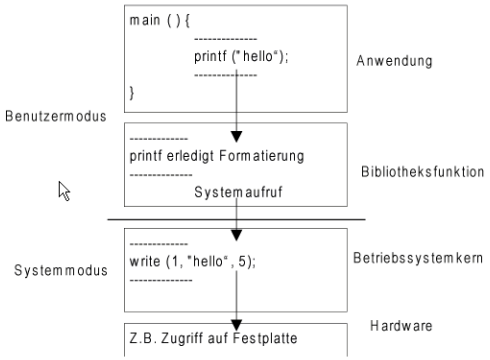
Linux Betriebssystem

Systemaufrufe

Virtuelle Maschine

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.

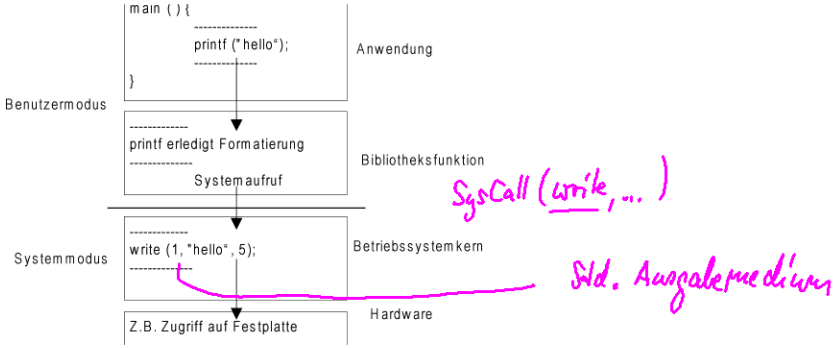
Beispiele von Systemaufrufen

Prozessmanagement: fork, waitpid, exit.

Dateimanagement: open, close, read, write.

Verzeichnismanagement: mkdir, rmdir, link, mount.

Systemaufrufe



Systemaufruf führt zum Übergang vom Benutzermodus in den Systemmodus.

Beispiele von Systemaufrufen

Prozessmanagement: fork, waitpid, exit.

Dateimanagement: open, close, read, write.

Verzeichnismanagement: mkdir, rmdir, link, mount.

Gerätemanagement: request/release device, get/set device attributes.

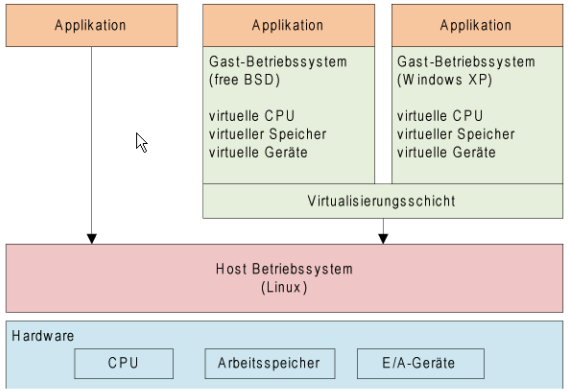
Kommunikationsmanagement: send/receive messages, create/delete connection.

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.

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

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.

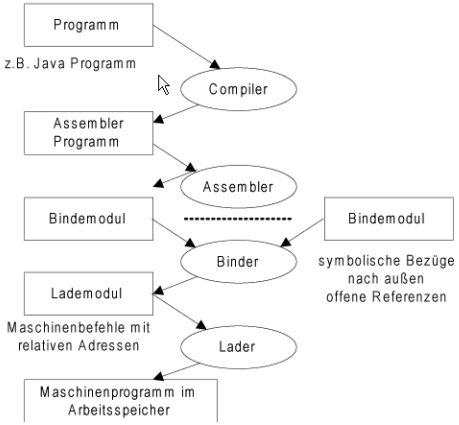
Verwendung von Namen für Adressen und Operationen.

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.

Programmieren auf der Assemblerschnittstelle; Verwendung von Assembler-Programmen

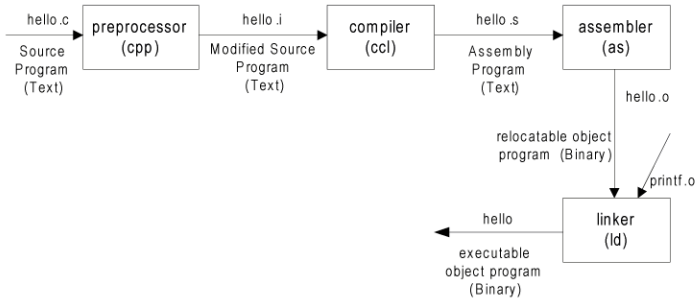


Beispiel Unix Compilersystem

Beispiel Unix Compilersystem

Aufruf:

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



Transformation in 4 Phasen

Preprocessor: Aufbereitung.

Compiler: Übersetzer von C nach Assembler.

Assembler: Übersetzer von Assembler nach Maschinensprache; Generierung des Objektprogramms.

Linker (Binder): Nachbearbeitung und Kombinationen verschiedener Module.

Hardwarenahe Programme

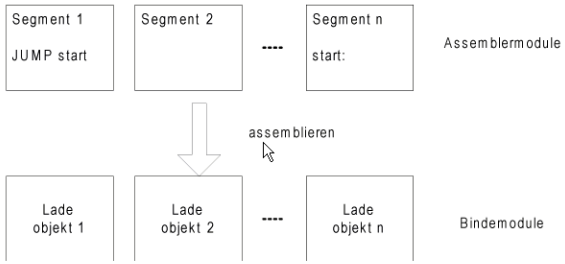
Nutzung von Assemblerprogrammen, aber auch C-Programmen zur Realisierung hardwarenaher Programme für ein Rechensystem.

Definitionen

Programmaufbereitung

Binder und Lader

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



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

Externe Referenzen

In einem Modul M_i können Referenzen auf Daten/Symbole auftreten, wobei die Daten/Symbole in einem anderen Modul M_j definiert werden. Beispiel: Symbol ***start*** wird in Segment 1 verwendet, und erst in Segment n definiert.

Behandlung externer Referenzen

Behandlung externer Referenzen

Für jede externe Referenz erzeugt der Assembler Informationen, die es dem Binder ermöglicht, aus den Einzelmoduln 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.

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

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.

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.

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.

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

Datenstrukturen eines Binde/Laders

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

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.

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.

Datenstrukturen eines Binde/Laders

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

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

Dynamisches Binden

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.

Grundlagen: Betriebssysteme und Systemsoftware (GBS)

- Prof. J. Schlichter
  - Lehrstuhl für Angewandte Informatik / Kooperative Systeme, Fakultät für Informatik, TU München
  - Boltzmannstr. 3, 85748 Garching

    Email: schlichter@in.tum.de

    Tel.: 089-289 18654

    URL: http://www11.informatik.tu-muenchen.de/