Konzepte von Betriebssystem-Komponenten

Programmstart & dynamische Bibliotheken

SS 05

Wladislaw Eckhardt

Wladi23@gmx.net

1 Einleitung

- Programmteile werden modularisiert
- Gemeinsamkeiten werden ausgelagert
- Wiederverwendbarkeit
- Bsp., Standartfunktionen von Programmiersprachen
- Standardfunktionen übersetzt und in Bibliotheken abgelegt
- statische Linken

Probleme durch statisches Linken ...

- Redundanz der Standardfunktionen
- Speicherplatzverschwendung
  - Arbeitsspeicher
  - Festplattenspeicher
- fehleranfällige Programmierung durch identischen Code
- Laufzeitverzögerung
- Updateprobleme

Motivation

- Maschinencode dürfen Programme nicht verändern
- Systeme unterbinden dies sowieso (ROM)
- Konstanten werden ebenfalls gemeinsam genutzt

Lösungsansatz :

Referenz auf gemeinsam genutzten Code verwenden

Mechanismen:

- Verwendung dynamische Bibliotheken oder ähnlicher Datenstruktur

UNIX: shared objects (abgekürzt so)

Windows: dynamic link library (DLL)

2 Bibliotheken

- Def.: Bibliotheken in der Informationsverarbeitung sind Sammlungen von widerverwendbaren Funktionsmodulen, dadurch ist klare Abkapselung und Strukturierung von dem eigentlichen Programm möglich.
  - Klassifizierung nach Funktionsart
  - Einteilung nach Programmiersprache
  - Weitere Unterscheidung, wann sie zu einem Programm hinzu gebunden werden

Zeitpunktklassen:

1. direkt nach Programmierübersetzung
2. bei jedem Programmstart
3. zu einem beliebigen Zeitpunkt

Statische Bindung - Bibliotheken

- Bindung zur Entwicklungszeit
- Programm + Bibliothek ein Objekt
- Bibliotheksdatei anschließend überflüssig
- Statische Linken
- Bibliotheken sind Archive
- Unflexibel ...

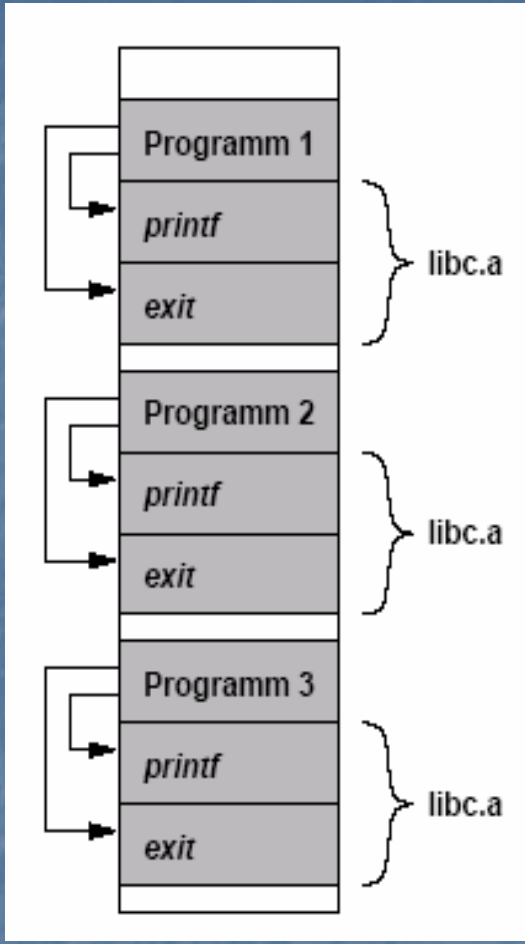


Abb. 1: statische Binden

Bibliotheken werden mit einem Programm in einer Datei gebunden:

Vorteile:

- immer alle Funktionen vorhanden

Nachteile:

- Längere Ladezeiten,
- Programme sind größer
- schlechte Erweiterbarkeit

Abb. 2: Objekte

Deshalb verwendet man diesen Mechanismus sehr selten, wenn man z. B. Programm fest mit Bibliothek verbunden haben möchte.

- **Dynamische Bindung zum Programmstart:**
  - Programm beim Start mit der passenden Bibliothek verbinden
  - dynamisches Binden
  - Bibliothek muss separat geladen werden
  - komplizierter Vorgang
  - gemeinsam nutzbare Bibliothek (shared libraries)
  - Aber dennoch wird dieser Mechanismus nicht Dll (dynamically linked libraries) genannt, da dieser Begriff dem nächsten zu Erläuternden verfahren gehört.

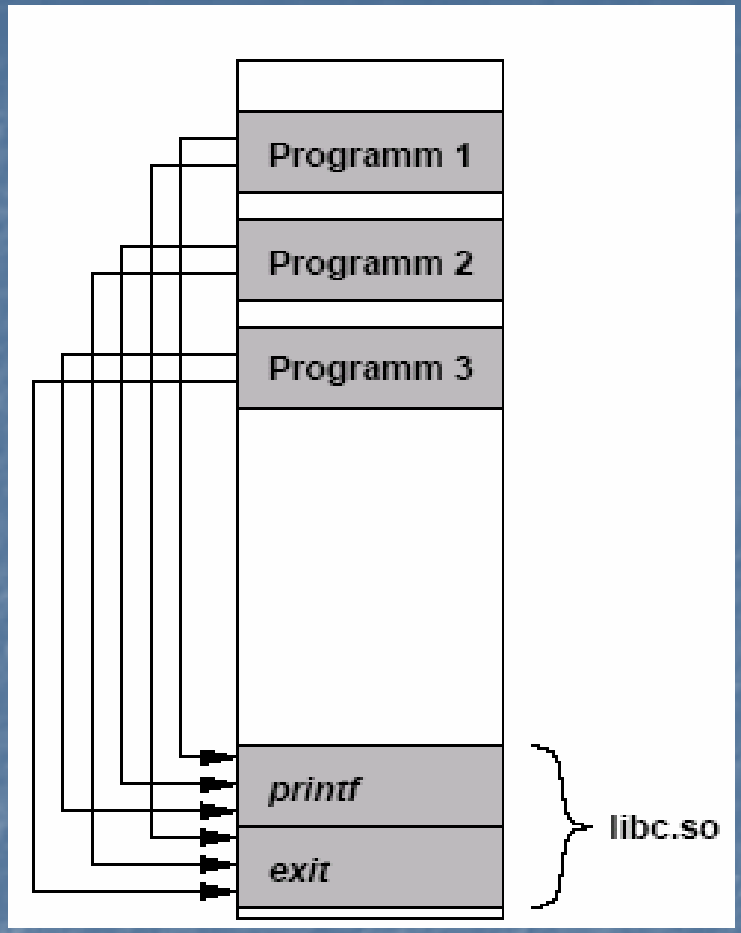


Abb. 3: gemeinsam genutzte Bibliotheken

Dynamische Bindung bzw. Bibliotheken

- Programm lädt Bibliotheken erst zu Laufzeit
- hohe Flexibilität
- Programm kann funktionieren auch wenn nicht alle Bib. vorhanden sind
- Bibliotheken werden erst dann geladen wenn sie gebraucht werden
- sehr kompliziert zu realisieren
- konsequentes Programmierkonzept notwendig
- unter Unix shared libraries genannt

Dynamische Bibliotheken

- Eine DLL ist eine Bibliothek, die zur Laufzeit geladen und mit dem Programm verbunden (gelinkt) wird

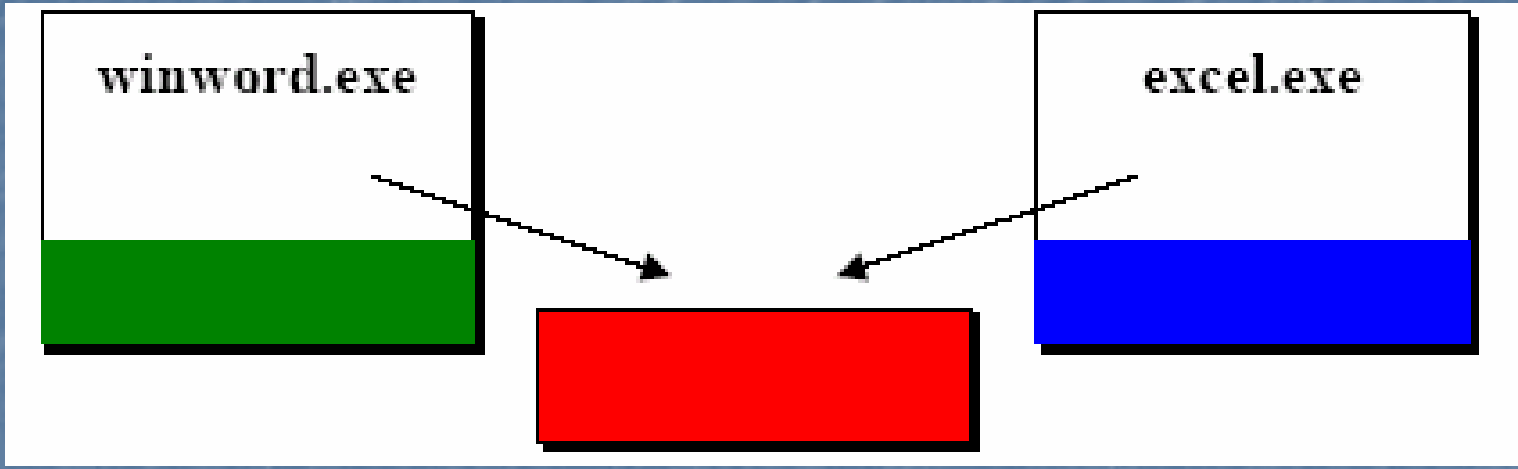
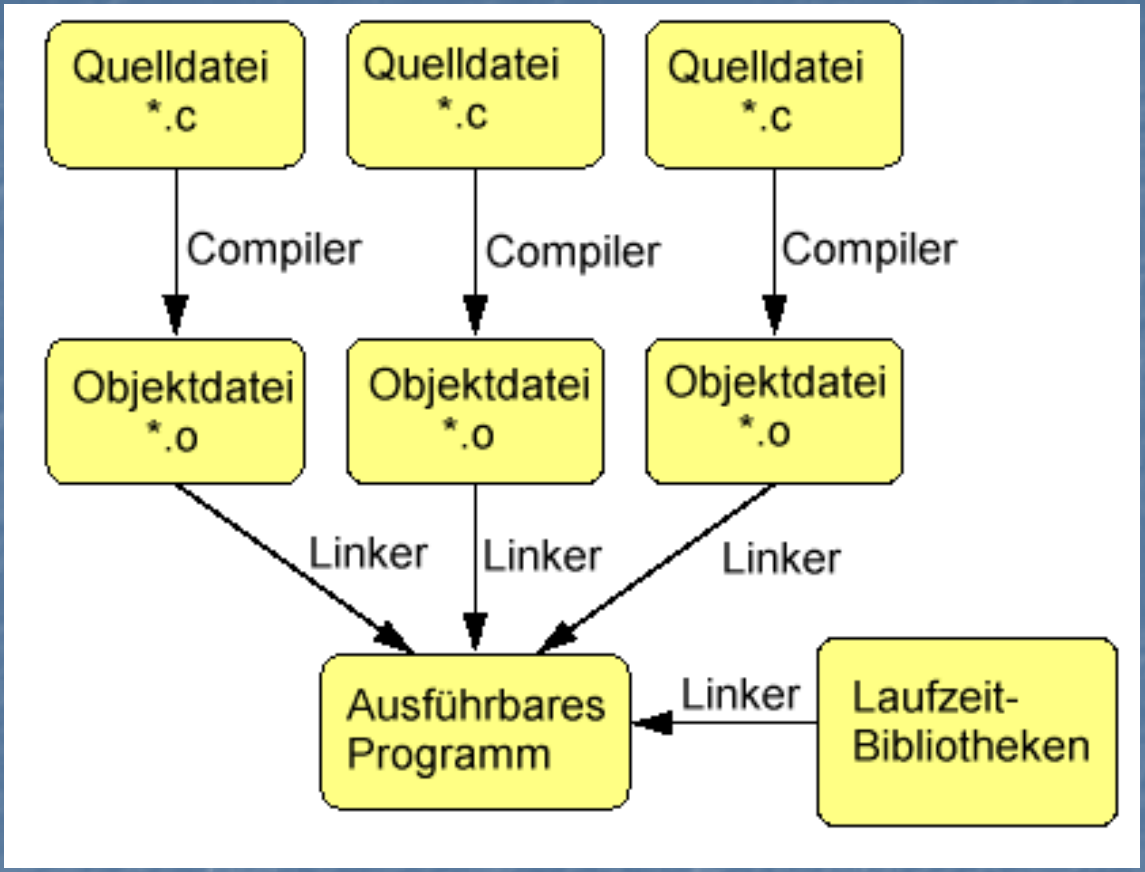


Abb. 4: Zugriff auf Bibliothek zur Laufzeit



- *Abb. 5: allgemeine Programmerzeugung*

3 Programmstart & Programmerzeugungsprozess

1. Der Präprozessor bearbeitet die Präprozessor-derektiven im Quelltext z. B. #include -oder #define Anweisungen

2. Der Compiler erzeugt ein Objektmodul

3. Der Linker erzeugt aus einem oder mehreren Objektmodulen ein lauffaehiges Programm. Die Objektmodule liegen entweder als einzelne Dateien (Suffix: *.o) oder in Form von Objektmodul-Bibliotheken (lib ...) vor

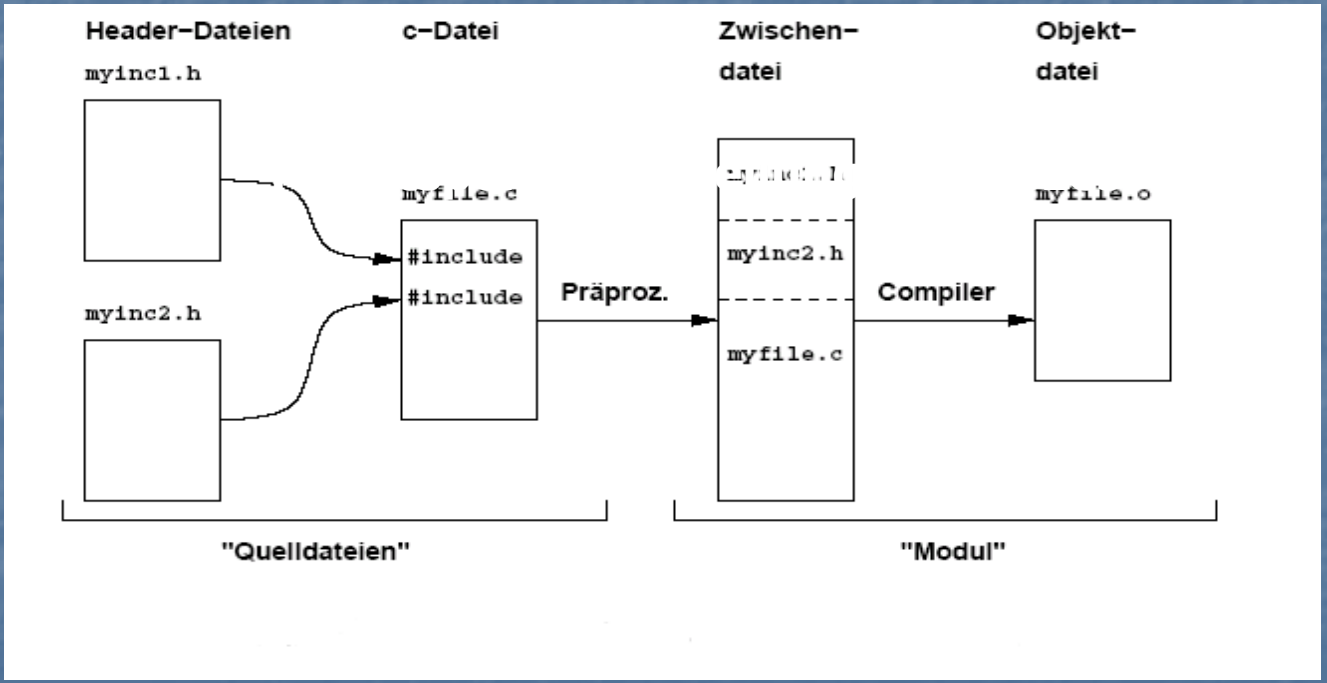


Abb. 6: Generierung eines Objektmoduls

Hauptspeichereinteilung für Prozesse

.code Segment (Programm)

.data Segment (statische Daten)

.bss Segment (dynamische Daten)

Stack

Abbildung 5: Einteilung des Hauptspeichers fuer Prozesse

4 Strategien zur Verwendung dynamischer Bibliotheken

festes Linken

- Dummy-Objekte werden statt Funktionen dazugelinkt
- festes Linken != statische Linken

dynamische Laden

- Funktionen durch BS-Funktionen nachgeladen

Mechanismus zur Realisierung der gemeinsam genutzten Bibliotheken

- Wo liegen eigentlich die Vorteile gegenüber der statischen Bindung?
- Anstelle der Funktionen (oder Daten) der Bibliothek werden beim Linken leere Dummy-Objekte dazugelinkt, um für den Linker die Referenzen zu erfüllen, diese bringen nur den Namen einer Funktion in die Objektdatei.
- Beim Start des Programms wird vom Laufzeitsystem automatisch die Bibliothek geladen.
- Referenzen auf die Dummys werden ersetzt
- möglicherweise müssen Bibliotheken nachgeladen werden
- interne Optimierung bleibt nach außen Verborgen
- sehr kompliziert zu realisieren ...

5 Beispiele

Erstellung statischer Bibliotheken:

ar <Schalter> <Libname> <Objektdateien>

ar cr **libmylib.a modul1.o modul2.o modul3.o**

Erstellung **dynamischer Bibliotheken:**

Nicht alle Unix-Systeme beherrschen den komplizierten Mechanismus des dynamischen Linken, auch die Implementierungen können sich unterscheiden.

Um dynamische Bibliotheken zu erstellen, müssen zunächst die Quelltextdateien mit dem zusätzlichen Schalter „-fPIC“ zu Objekdateien übersetzt werden.

Beispiel:

gcc –g –fPIC –c modul1.c

gcc –g modul1.o –shared –Wl, soname, libirgendwas.so.1 –o libirgendwas.so.1.0

Zusammenfassung

Vorteile dynamischer Bibliotheken gegenüber statischen Bibliotheken:

- *Einsparung am Speicher -> Geschwindigkeitsvorteile*
- *schnelleren Programmen*
- *kleineren Programmen*
- *Fehler in Programmen können schnell behoben werden*
- *ausführbare Code einer dynamischen Bibliothek wird von System nur einmal in den Speicher geladen*
- Updates erfordern relativ wenig Aufwand

Nachteile dynamischer Bibliotheken gegenüber statischen:

- Ein mit dynamischen Bibliotheken gelinktes Programm ist für sich allein nicht ablauffähig, da es immer die zugehörigen dynamischen Bibliotheken benötigt

- Dynamischen Bibliotheken müssen beim Programmstart erst gesucht und geladen

Fragen ?