

### E.1 Übersetzen - Objektmodule

#### ■ 1. Schritt: Präprozessor

- ◆ entfernt Kommentare, wertet Präprozessoranweisungen aus
    - ▶ fügt include-Dateien ein
    - ▶ expandiert Makros
    - ▶ entfernt Makro-abhängige Code-Abschnitte (*conditional code*)
- Beispiel:

```
#define DEBUG
...
#ifndef DEBUG
    printf("Zwischenergebnis = %d\n", wert);
#endif DEBUG
```

- ◆ Zwischenergebnis kann mit `cc -P datei.c` als `datei.i` erzeugt werden oder mit `cc -E datei.c` ausgegeben werden

### E.1 Übersetzen - Objektmodule (2)

#### ■ 2. Schritt: Compilieren

- ◆ übersetzt C-Code in Assembler
- ◆ Zwischenergebnis kann mit `cc -S datei.c` als `datei.s` erzeugt werden

#### ■ 3. Schritt: Assemblieren

- ◆ assembliert Assembler-Code, erzeugt Maschinencode (Objekt-Datei)
- ◆ standardisiertes Objekt-Dateiformat: ELF (Executable and Linking Format) (vereinfachte Darstellung) - in nicht-UNIX-Systemen andere Formate
  - ▶ Maschinencode
  - ▶ Informationen über Variablen mit Lebensdauer static (ggf. Initialisierungswerte)
  - ▶ Symboltabelle: wo stehen welche globale Variablen und Funktionen
  - ▶ Relokierungsinformation: wo werden welche "nicht gefundenen" globalen Variablen bzw. Funktionen referenziert
- ◆ Zwischenergebnis kann mit `cc -c datei.c` als `datei.o` erzeugt werden

## E.2 Binden und Bibliotheken

### ■ 4. Schritt: Binden

- ◆ Programm `ld` : ( *linker* ), erzeugt ausführbare Datei ( *executable file* )
  - ▶ ebenfalls ELF-Format (früher a.out-Format oder COFF)
- ◆ Objekt-Dateien (.o-Dateien) werden zusammengebunden
  - ▶ noch nicht abgesättigte Referenzen auf globale Variablen und Funktionen in anderen Objekt-Dateien werden gebunden (Relokation)
- ◆ nach fehlenden Funktionen wird in Bibliotheken gesucht

#### ■ statisch binden

- ◆ alle fehlenden Funktionen werden aus Bibliotheken genommen und in die ausführbare Datei einkopiert
  - ▶ ausführbare Datei ggf. sehr groß
- ◆ Funktionen die in vielen Programmen benötigt werden (z. B. `printf`) werden überall einkopiert

### E.2 Binden und Bibliotheken (2)

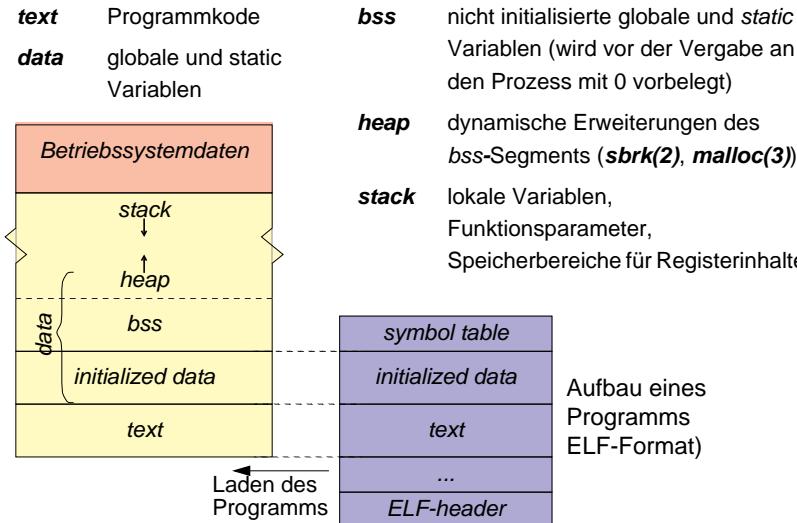
#### ■ dynamisch binden

- ◆ Funktionen in gemeinsam nutzbare Bibliotheken (*shared libraries*) werden nicht in die ausführbare Datei einkopiert
  - ▶ ausführbare Datei enthält weiterhin nicht-relokierte Referenzen
  - ▶ ausführbare Dateien sind kleiner, mehrfach genutzte Funktionen sind nur einmal in der shared library abgelegt
  - ▶ Relokation erfolgt beim Laden

## E.3 Programme und Prozesse

- **Programm:** Folge von Anweisungen  
(hinterlegt beispielsweise als ausführbare Datei auf dem Hintergrundspeicher)
- **Prozess:** Programm, das sich in Ausführung befindet, und seine Daten  
(Beachte: ein Programm kann sich mehrfach in Ausführung befinden)
  - eine konkrete Ausführungsumgebung für ein Programm  
Speicher, Rechte, Verwaltungsinformation (verbrauchte Rechenzeit,...),...
- jeder Prozess bekommt einen eigenen virtuellen Adressraum zur Verfügung gestellt
  - eigener (virtueller) Speicherbereich von 0 bis 2 GB (oder mehr bis 4 GB)
  - Datenbereiche von verschiedenen Prozessen und Betriebssystem sind gegeneinander geschützt
  - Datentransfer zwischen Prozessen nur durch Vermittlung des Betriebssystems möglich

## E.4 Speicherorganisation eines Prozesses



## E.4 Speicherorganisation eines Prozesses (2)

- Abbildung des virtuellen Adressraums in den realen Hauptspeicher durch Seitenadressierung (**Paging**)
  - Adressraum ist in kleine (4 oder 8 kB) Stücke unterteilt (**Seiten**)
  - jede Seite wird über eine Tabelle in ein entsprechendes Stück des Hauptspeichers (**Kachel**) abgebildet
  - bei jedem Speicherzugriff wird die virtuelle Adresse in die entsprechende physikalische Adresse umgerechnet (spezielle Hardware: Memory Management Unit - MMU)
  - zu jeder Seite sind Zugriffsrechte vermerkt (nur lesen, lesen+schreiben, Maschinenbefehle ausführen)
  - eine Seite kann bei Speichermangel von Betriebssystem auf Festplatte ausgelagert werden und bei Bedarf automatisch wieder eingelagert werden

## E.4 Speicherorganisation eines Prozesses(2)

