Imperative Speicherorganisation

und Behandlung von Prozeduraufrufen

Generelle von-Neumann Sicht

- Einem Pogramm gehört der gesamte Speicher
- Der Speicher ist in Zellen adressiert
- Ein Teil des Speichers ist für den Programmcode und weiteres für das Gesamtprogramm nötige reserviert
- Der Rest - der *freie Speicher* - ist dynamischer Speicher. Er wird nach zwei Methoden vergeben, und der Speicher dafür wird von den Rändern zu Mitte vergeben: *stack* und *heap*.

Stack

- Zur Speicherung von lokalen Variablen und Zusatzinformationen zur Behandlung eines Funktionsaufrufs verwendet man den Stack.
- *Stackframes* oder *Activation Records (FSB, ...)* werden für jeden Prozeduraufruf (ggf. auch für jeden Blockmit Ílokalen Variablen) auf den Stack gelegt. In ihnen steckt die nötige Information für die Abwicklung des Prozeduraufrufs, und zum aufzuräumen und der Herstellung der alte Situation, ergänzt um eventuelle Rckgabewerte und mit einem vorgerückten *program counter*.

Stack

Der Stack wächst und schrumpft nur an einem Ende, an der Adresse

stack pointer (sp)

zentrale Operationen sind

push = Wert auf den Stack legen, sp erhöhen
pop = Wert zurückliefern, sp vermindern

heap

- Dynamische Array sind ggf. noch auf dem Stack verwaltbar - ihre Grösse ist ja mindestens zur Aufrufzeit einer Prozedur bekannt. Andere Grössen sind auch dann noch unkalkulierbar, sie müssen zur Laufzeit mittels entsprechender Hilfsprozeduren auf dem sog. *heap* reserviert (*allocate*) und wieder frei gegeben (*free*) werden

Basisgrössen

Der Speicherbedarf von Basistypen (integer, float, boolean, character) ist bekannt. Wir gehen hier von der Grundgrösse 1 Byte = 8 Bit aus, und gehen bei den Basistypen von einfachen Vielfachen dieser Grösse aus.

Je nach vorhandener realer Speicherarchitektur ist dabei ggf. eine Ausrichtung auf bestimmte Speicheradressen zu beachten, z.B. bei 32Bit-Speicherwörtern auf eine 4-Byte Ausrichtung, und entsprechend zu füllen.

Umgang mit Prozeduren

Der Code von Prozeduren ist nur ein mal im Speicher vorhanden. Die für einen Aufruf gültigen lokalen Werte müssen dahingegen ggf. mehrfach verfügbar sein.

In den meisten Sprachen sind sich rekursiv aufrufende Prozeduren erlaubt, dafür benötigt man eine effiziente Behandlung der Zugriffe auf die richtigen Variablen- bzw. Parameterwerte über die dynamische Verweiskette.

Umgang mit Prozeduren

Ist es erlaubt, Prozedurdeklarationen zu schachteln, hat man zur Bestimmung der korrekten Namensbindung neben der dynamischen auch noch die statische Verweiskette zu beachten:

Zur Bestimmung der passende Deklaration zu einem Auftreten eines Variablennamens sucht man in der (statischen) Verweiskette aufsteigend nach der ersten gültigen Deklaration.

Umgang mit Prozeduren - scope

Eine Variable ist

- *sichtbar* in dem scope, in dem sie deklariert wurde, und allen eingeschlossenen scopes, in denen keine Variable gleichen Namens deklariert wurde.
- *gültig* in dem scope, in dem sie deklariert wurde, und allen eingeschlossenen scopes.
- *verdeckt* in den von dem deklarierenden scope eingeschlossenen scopes, in denen eine Variable gleichen Namens deklariert wurde.

Umgang mit Prozeduren
Activation Records

Um die Speicherbedarf von Prozeduren zu handhaben verwendet man den Stack, wobei zusätzliche Verwaltungsinformationen für jeden Aufruf hinzukommen.

Benötigt werden mehrere Stackzellen.

Man bezeichnet diesen Bereich auf dem Stack als *Activation Record* (AR) oder *Stack Frame*, auch *Display* kommt vor

Die Aufrufverwaltung kann über Rücksprungketten in den ARs erfolgen, oder durch eine gesonderte Display-Technik mit Hilfe von (Index)Registern. Letztere ist effizienter.

Informationen in Activation Records

Welche Information muss in einem AR verwaltet werden?

Informationen in Activation Records

Welche Information muss in einem AR verwaltet werden?

Das hängt natürlich auch an der Rechnerarchitektur, manche Architekturen unterstützen direkt einige der Techniken, anderer weniger.
Insofern folgt hier ein recht vollständiges Beispiel für MMS und eine sehr einfache Rechnerarchitektur.

Informationen in Activation Records

Welche Information muss in einem AR verwaltet werden?

Prinzipiell muss man alle Werte, die man zur Abwicklung eines Aufrufs braucht, erst retten, um am Ende den vorherigen Zustand sauber wiederherstellen zu können.
Hier: MDL, BFS, PC, IR[stat. Niveau]

(IRs dienen zur Korrelation der statische Verwieskette mit der Aufrufkette.)

Informationen in Activation Records

Welche Information muss in einem AR verwaltet werden?

1. Ein Zeiger auf den AR des dynamischen Vorgängers (*Rettung altes MDL*)
2. das statische Niveau des stat. Vorgängers (bei Sprachen mit Funktions/Deklarationsschachtelung).
3. Alter Wert aus IR[stat. Niveau] (Zum Wiederherstellen der statischen Verweiskette)

...

Informationen in Activation Records

Welche Information muss in einem AR verwaltet werden?

...

4. ProgramCounter+1 (Rückkehradresse)

5 . BFS - Ein Zeiger auf den Beginn des freien Speichers hinter dem nun folgenden dynamischen Teil des ARs. (Ggf. durch SP / MDL ersetzbar)

...

Informationen in Activation Records

Welche Information muss in einem AR verwaltet werden?

...

Der Platzbedarf für 1-5 liegt fest, es kann also alles mit Hilfe eines Zeigers auf den Beginn des ARs korrekt adressiert werden.
Zellen 1-5 nennt man auch *Prozedur Linkage.*

...

Informationen in Activation Records

Welche Information muss in einem AR verwaltet werden?

...

6. Aktuelle Parameter (ggf. auch über Register)
7. Platz für lokale Variablen
8. Platz für temporäre lokale Variablen
9. Platz für dynamische Array (soweit die Sprache das zulässt)

Umgang mit ARs - Prozeduraufruf

Mit dem gerade beschriebenen AR würde bei einem Prozeduraufruf folgendes passieren:

1. Stor[BFS+0] := MDL; MDL := BFS
2. Stor[BFS+1] := stat. Niveau der aufgeruf. Prozedur - 1
3. Stor[BFS+2] := IR[stat. Niveau der aufgeruf. Prozedur]; IR[stat. Niveau der aufgeruf. Prozedur] := MDL
4. Stor[BFS+3] := PC; PC := Code der Prozedur
5. Stor[BFS+4] := BFS + 5 + Platz für Var. und Param.
6. Stor[BFS+5 : BFS+N] mit akt. Parametern füllen
7. BFS := Stor[BFS+4]
8. Mit der Programmausführung fortfahren

Umgang mit ARs - Prozedurende

Folgendes passiert bei Ende einer Prozedur:

1. PC := Stor[MDL+3]
2. BFS := MDL
3. IR[stat. Niveau] := Stor[MDL+2]
4. MDL := Stor[MDL]
5. ggf. Stor[BFS] := Rückgabewert
6. Setze Ausführung bei PC fort.

Umgang mit ARs - Variablenzugriff

Wie findet das Verfahren die richtigen Variablen?

- lokale Variablen oder aktuelle Parameter sind in **Stor [MDL+n]**, n>4, und der Compiler kennt n.
- zur aktuellen Prozedur globale Variablen haben ein statisches Niveau N. Der Wert steht dann in **Stor[IR[N] +n]**, n wie oben.

Umgang mit Prozeduren
Activation Records

Beispiel

für ein AR (dort genannt *Festspeicherblock*, *FSB*) in der Realisation bei MMS (LIppe)

mit Display-Technik in Indexregistern:

Material/AbbSpeicher2.png

Lernziele

- wissen, wie Prozedurcalls zur Laufzeit gehandhabt werden
- Die Bedeutung der Werte in der *prozedur linkage* kennen und erklären können
- Statische und dynamische Verweiskette erläutern können.