Compilerkonstruktion

Wintersemester 2015/16

Prof. Dr. R. Parchmann

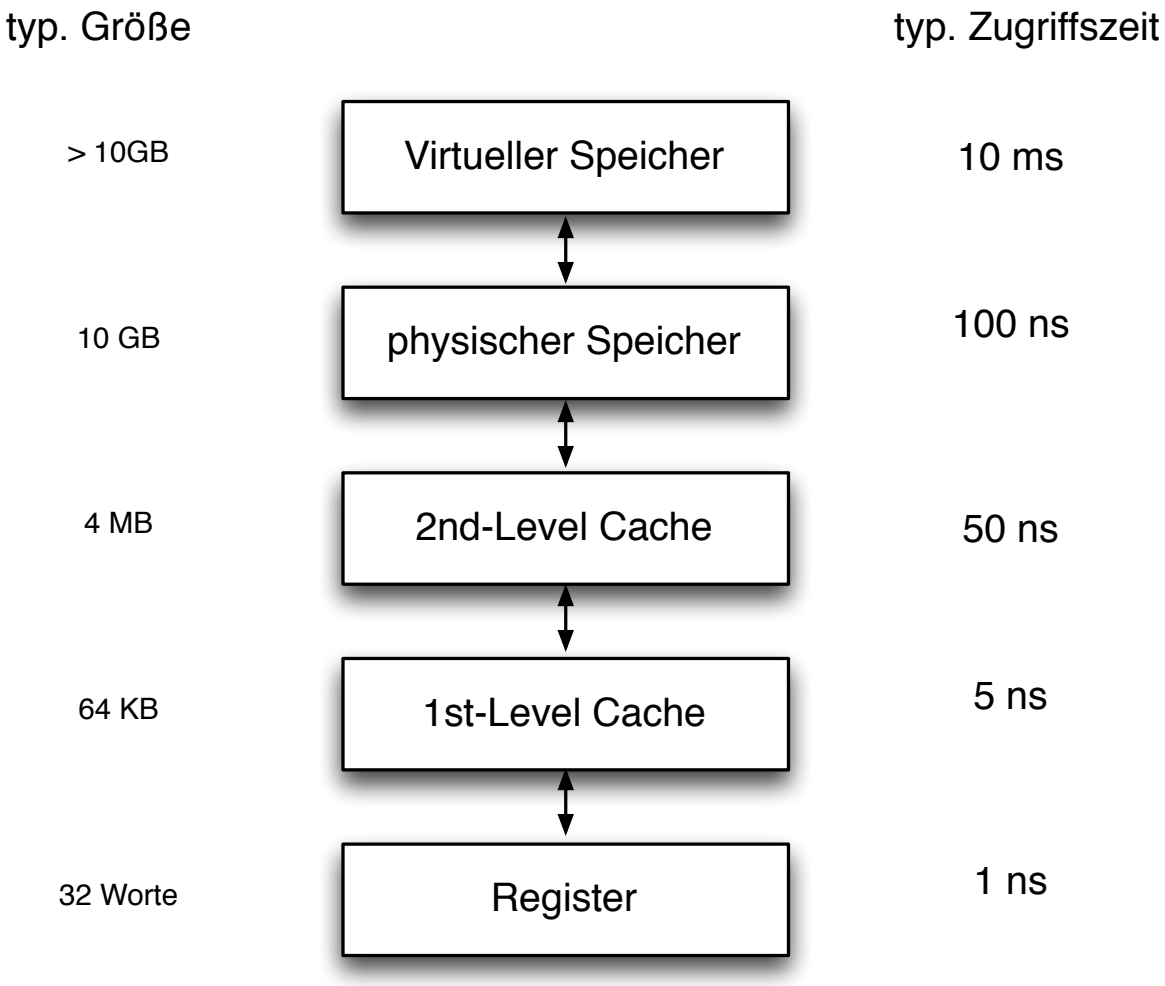
17. November 2015

Heap Management

Der Heap-Speicher eines Programms wird hauptsächlich zum Speichern von Daten und Objekten benutzt, die zur Laufzeit des Programms erzeugt und später wieder vom Programm freigegeben werden. Die Speicheranforderung geschieht etwa über eine `new`-Anweisung in einer Prozedur, um Speicherplatz für ein neues Datenobjekt anzufordern (Konstruktor). Dieses Datenobjekt existiert aber auch noch, wenn die Prozedur beendet wird.

In einigen Sprachen muss der Programmierer selbst dafür Sorge tragen, den Speicherplatz wieder frei zu geben (etwa in in C oder C++), in anderen Sprachen gibt es ein Garbage-Collector Verfahren, das den Heap nach nicht mehr benötigten Datenobjekten durchsucht und diese wieder dem freien Speicher zuordnet.

Die Speicherhierarchie eines Rechners



- Der Transfer zwischen den verschiedenen Speicherstufen findet immer in größeren Einheiten statt, beim virtuellem Speicher vom Betriebssystem gesteuert in etwa 32 KB großen Blöcken, bei den Cache-Speichern von der Hardware gesteuert in kleineren Blöcken von etwa 256 Bytes.

- Der Transfer zwischen den verschiedenen Speicherstufen findet immer in größeren Einheiten statt, beim virtuellem Speicher vom Betriebssystem gesteuert in etwa 32 KB großen Blöcken, bei den Cache-Speichern von der Hardware gesteuert in kleineren Blöcken von etwa 256 Bytes.
- Die Speicher-Hardware arbeitet üblicherweise mit einer Varianten des *most recently used*-Algorithmus, das bedeutet, dass versucht wird, die zuletzt benutzen Speicherbereiche im Cachespeicher zu behalten.

- Der Transfer zwischen den verschiedenen Speicherstufen findet immer in größeren Einheiten statt, beim virtuellem Speicher vom Betriebssystem gesteuert in etwa 32 KB großen Blöcken, bei den Cache-Speichern von der Hardware gesteuert in kleineren Blöcken von etwa 256 Bytes.
- Die Speicher-Hardware arbeitet üblicherweise mit einer Varianten des *most recently used*-Algorithmus, das bedeutet, dass versucht wird, die zuletzt benutzen Speicherbereiche im Cachespeicher zu behalten.
- Es ist wichtig, das Programm so zu schreiben (und zu übersetzen!), dass man größtmögliche Lokalität erreicht.

- Der Transfer zwischen den verschiedenen Speicherstufen findet immer in größeren Einheiten statt, beim virtuellem Speicher vom Betriebssystem gesteuert in etwa 32 KB großen Blöcken, bei den Cache-Speichern von der Hardware gesteuert in kleineren Blöcken von etwa 256 Bytes.
- Die Speicher-Hardware arbeitet üblicherweise mit einer Varianten des *most recently used*-Algorithmus, das bedeutet, dass versucht wird, die zuletzt benutzen Speicherbereiche im Cachespeicher zu behalten.
- Es ist wichtig, das Programm so zu schreiben (und zu übersetzen!), dass man größtmögliche Lokalität erreicht.
- 80 -20 Regel: 80% der Rechenzeit wird in 20% des Maschinencodes verbraucht wird.

Speicherverwaltung

Die Speicherverwaltung für den Heap-Speicher muss zwei wichtige Aufgaben erfüllen:

Allocation Wenn ein Programm Speicherplatz für ein Objekt anfordert, muss die Speicherverwaltung ein genügend großes zusammenhängendes Stück aus dem Heap-Speicher finden und eine Referenz auf diesen Speicherbereich zurückgeben.

Deallocation Die Speicherverwaltung muss Speicherbereiche, die vom Anwendungsprogramm nicht mehr benötigt werden, wieder zum Bereich des freien Speichers zuordnen.

Die Speicherverwaltung sollte diese Aufgaben schnell und effizient erfüllen und dabei selbst nicht zu viele Ressourcen verbrauchen.

Manuelle Allokation und Deallokation

- Zur Allokation werden häufig Varianten des Best-Fit oder First-Fit Verfahrens angewendet.

Manuelle Allokation und Deallokation

- Zur Allokation werden häufig Varianten des Best-Fit oder First-Fit Verfahrens angewendet.
- Bei der Deallokation muss nur darauf geachtet werden, dass nebeneinander liegende freie Bereiche des Speichers wieder zu einem größeren Bereich verschmolzen werden. (Verminderung der Fragmentierung des freien Speichers)

Manuelle Allokation und Deallokation

- Zur Allokation werden häufig Varianten des Best-Fit oder First-Fit Verfahrens angewendet.
- Bei der Deallokation muss nur darauf geachtet werden, dass nebeneinander liegende freie Bereiche des Speichers wieder zu einem größeren Bereich verschmolzen werden. (Verminderung der Fragmentierung des freien Speichers)

Nachteil: Bei ungenauer Programmierung kann man eventuell vergessen, nicht mehr benötigte Speicherbereiche zurückzugeben (*memory leak*) oder man referenziert bereits zurückgegebene Speicherbereiche (*dangling-pointer*).

Folge:

- erhöhter Speicherbedarf und so eventuell nach einiger Laufzeit Abbruch des Programms wegen Speichermangel
- das Programm liefert fehlerhafte Daten oder endet mit einer Exception. Dieser Fehler ist häufig schlecht reproduzierbar und schwer zu lokalisieren.

Reference Counting

- Jedes dynamisch allokiertes Objekt wird durch einen Zähler (*reference count*) erweitert.

Reference Counting

- Jedes dynamisch allokiertes Objekt wird durch einen Zähler (*reference count*) erweitert.
- Bei einer neuen Referenz zu einem Objekt muss der Zähler inkrementiert werden.

Reference Counting

- Jedes dynamisch allokiertes Objekt wird durch einen Zähler (*reference count*) erweitert.
- Bei einer neuen Referenz zu einem Objekt muss der Zähler inkrementiert werden.
- Wird eine Referenz gelöscht, muss der Zähler dekrementiert werden.

Reference Counting

- Jedes dynamisch allokiertes Objekt wird durch einen Zähler (*reference count*) erweitert.
- Bei einer neuen Referenz zu einem Objekt muss der Zähler inkrementiert werden.
- Wird eine Referenz gelöscht, muss der Zähler dekrementiert werden.
- Erreicht der Reference Count eines Objektes den Wert 0, kann das Objekt deallokiert werden.

Reference Counting

- Jedes dynamisch allokiertes Objekt wird durch einen Zähler (*reference count*) erweitert.
- Bei einer neuen Referenz zu einem Objekt muss der Zähler inkrementiert werden.
- Wird eine Referenz gelöscht, muss der Zähler dekrementiert werden.
- Erreicht der Reference Count eines Objektes den Wert 0, kann das Objekt deallokiert werden.
- Enthält das Objekt Referenzen zu anderen Objekten, so müssen deren Referenz Counts ebenfalls dekrementiert werden.

Diese Methode hat Probleme bei zirkulären Strukturen, ausserdem kostet sie bei jeder Operation, die eine Referenz verändert.

Garbage Collection

- Grundidee: Objekte, die nicht über Referenzen erreichbar sind, können freigegeben werden.
- Voraussetzung: die Programmiersprache ist Typ-sicher. (für jede Variable ist der Typ des Objekts auf das sie verweist, zumindest zur Laufzeit bekannt.)
- man muss wissen, welche Teile des Objekts wiederum Referenzen auf andere Objekte enthalten.

Das ist problematisch für Sprachen, in denen Pointer-Arithmetik zulässig ist und somit Pointer mitten in ein Objekt zeigen können.

Garbage Collection

Qualitätsmerkmale eines Garbage-Collector Verfahrens

- Das Verfahren sollte die Laufzeit des Anwendungsprogramms (des **Mutators**) nicht zu sehr verlangsamen.

Garbage Collection

Qualitätsmerkmale eines Garbage-Collector Verfahrens

- Das Verfahren sollte die Laufzeit des Anwendungsprogramms (des **Mutators**) nicht zu sehr verlangsamen.
- Das Verfahren sollte nicht zu viel Speicherplatz benötigen und die Fragmentierung des freien Speichers vermeiden.

Garbage Collection

Qualitätsmerkmale eines Garbage-Collector Verfahrens

- Das Verfahren sollte die Laufzeit des Anwendungsprogramms (des **Mutators**) nicht zu sehr verlangsamen.
- Das Verfahren sollte nicht zu viel Speicherplatz benötigen und die Fragmentierung des freien Speichers vermeiden.
- Da das Verfahren quasi parallel zum Anwendungsprogramm läuft, sollte es keine langen Pausen in der Abarbeitung des Anwendungsprogramms produzieren.

Garbage Collection

Qualitätsmerkmale eines Garbage-Collector Verfahrens

- Das Verfahren sollte die Laufzeit des Anwendungsprogramms (des **Mutators**) nicht zu sehr verlangsamen.
- Das Verfahren sollte nicht zu viel Speicherplatz benötigen und die Fragmentierung des freien Speichers vermeiden.
- Da das Verfahren quasi parallel zum Anwendungsprogramm läuft, sollte es keine langen Pausen in der Abarbeitung des Anwendungsprogramms produzieren.
- Das Verfahren sollte die Lokalität des Anwendungsprogramms verbessern oder zumindest nicht verschlechtern.

Erreichbarkeit von Objekten

- Die Basismenge der erreichbaren Objekte sind alle, die direkt, ohne Dereferenzierung eines Zeigers vom Programm aus erreicht werden können.
- Alle Objekte in der Basismenge sind direkt erreichbar.

Erreichbarkeit von Objekten

- Die Basismenge der erreichbaren Objekte sind alle, die direkt, ohne Dereferenzierung eines Zeigers vom Programm aus erreicht werden können.
- Alle Objekte in der Basismenge sind direkt erreichbar.
- Rekursiv ist jedes Objekt, dessen Referenz in einer Instanzvariablen eines erreichbaren Objekts gespeichert ist, ebenfalls erreichbar.

Probleme speziell bei optimierenden Compilern: Variablenwerte werden zeitweise nur in einem Register gehalten oder werden erst durch Addition eines Offsets zu einer korrekten Referenz.(*versteckte Referenzen*)

Zur Lösung dieser Probleme kann der Compiler:

- Garbage Collection nur zu gewissen Zeitpunkten erlauben, wenn diese „versteckten“ Referenzen nicht existieren.
- dem Garbage Collector zusätzliche Informationen geben oder
- er kann garantieren, dass sich alle erreichbaren Objekte aus der Basismenge rekursiv bestimmen lassen.

Es gibt vier Situationen, in denen der Mutator die Menge der erreichbaren Objekte verändert:

- Allokation von neuen Objekten.

Es gibt vier Situationen, in denen der Mutator die Menge der erreichbaren Objekte verändert:

- Allokation von neuen Objekten.
- Übergabe von Parameter und Rückgabewerten.

Es gibt vier Situationen, in denen der Mutator die Menge der erreichbaren Objekte verändert:

- Allokation von neuen Objekten.
- Übergabe von Parameter und Rückgabewerten.
- Zuweisen von Referenzen.

Es gibt vier Situationen, in denen der Mutator die Menge der erreichbaren Objekte verändert:

- Allokation von neuen Objekten.
- Übergabe von Parameter und Rückgabewerten.
- Zuweisen von Referenzen.
- Beenden von Prozeduren

Garbage Collection durch Reference Counting

Jedes Objekt hat ein zusätzliches Feld, das den Reference Count enthält.

- Bei Erzeugung eines Objekts wird dieser Zähler auf 1 gesetzt.

Garbage Collection durch Reference Counting

Jedes Objekt hat ein zusätzliches Feld, das den Reference Count enthält.

- Bei Erzeugung eines Objekts wird dieser Zähler auf 1 gesetzt.
- Wird ein Objekt als Parameter an eine Prozedur übergeben oder als Rückgabewert zurückgegeben, wird der Zähler inkrementiert.

Garbage Collection durch Reference Counting

Jedes Objekt hat ein zusätzliches Feld, das den Reference Count enthält.

- Bei Erzeugung eines Objekts wird dieser Zähler auf 1 gesetzt.
- Wird ein Objekt als Parameter an eine Prozedur übergeben oder als Rückgabewert zurückgegeben, wird der Zähler inkrementiert.
- Bei einer Zuweisung `a = b` von Referenzen muss der Zähler des Objekts b inkrementiert und des Objekts von a dekrementiert werden.

Garbage Collection durch Reference Counting

Jedes Objekt hat ein zusätzliches Feld, das den Reference Count enthält.

- Bei Erzeugung eines Objekts wird dieser Zähler auf 1 gesetzt.
- Wird ein Objekt als Parameter an eine Prozedur übergeben oder als Rückgabewert zurückgegeben, wird der Zähler inkrementiert.
- Bei einer Zuweisung `a = b` von Referenzen muss der Zähler des Objekts b inkrementiert und des Objekts von a dekrementiert werden.
- Beim Verlassen einer Prozedur müssen die Zähler aller Objekte, die über lokale Variable referenziert werden, dekrementiert werden.

- Wird der Reference Count eines Objekts auf 0 gesetzt, so müssen die Reference Counter alle Objekte, die über Instanzenvariablen dieses Objekts direkt erreichbar sind ebenfalls dekrementiert werden.

Nachteil dieser Methode:

- Der Overhead durch einzuführende extra Operationen ist relativ groß ist und nicht direkt von der Zahl der Objekte abhängig. Von Vorteil ist allerdings, dass das Garbage Collecting inkrementell stattfindet, so dass dieses Verfahren besonders bei zeitkritischen Anwendungen Vorteile hat.
- Bei zirkulären Strukturen, bei denen kein Mitglied in der Basismenge liegt, versagt diese Methode. Hier muss man besondere Vorkehrungen treffen.

Mark-and-Sweep Garbage Collector

Für jedes Objekt muss Platz für ein Markierungsbit reserviert werden.
Arbeitsweise:

- Das Anwendungsprogramm (der Mutator) allokiert Speicher für neue Objekte.

Mark-and-Sweep Garbage Collector

Für jedes Objekt muss Platz für ein Markierungsbit reserviert werden.
Arbeitsweise:

- Das Anwendungsprogramm (der Mutator) allokiert Speicher für neue Objekte.
- Nach einer gewissen Zeit beginnt der Garbage Collector die erreichbaren Objekte zur markieren.

Mark-and-Sweep Garbage Collector

Für jedes Objekt muss Platz für ein Markierungsbit reserviert werden.
Arbeitsweise:

- Das Anwendungsprogramm (der Mutator) allokiert Speicher für neue Objekte.
- Nach einer gewissen Zeit beginnt der Garbage Collector die erreichbaren Objekte zur markieren.
- Anschließend durchläuft der Garbage Collector den gesamten Heap und sammelt die nicht markierten Bereiche in einer Liste der freien Speicherbereiche,

Varianten dieses Verfahrens

- Der Garbage Collector führt in der Sweep-Phase eine Kompaktifizierung des Speichers durch, indem es alle erreichbaren Objekte an ein Ende des Heaps verschiebt.
- Beim Copying Garbage Collector Verfahren ist der gesamte Heap ist in zwei Hälften aufgeteilt.
  - Der Mutator arbeitet zunächst nur in einer Hälfte und allokiert dort Objekte, bis der freie Speicher in dieser Hälfte knapp wird.

Varianten dieses Verfahrens

- Der Garbage Collector führt in der Sweep-Phase eine Kompaktifizierung des Speichers durch, indem es alle erreichbaren Objekte an ein Ende des Heaps verschiebt.
- Beim Copying Garbage Collector Verfahren ist der gesamte Heap ist in zwei Hälften aufgeteilt.
  - Der Mutator arbeitet zunächst nur in einer Hälfte und allokiert dort Objekte, bis der freie Speicher in dieser Hälfte knapp wird.
  - Der Garbage Collector kopiert dann die erreichbaren Objekte in die andere Hälfte des Heaps.

Varianten dieses Verfahrens

- Der Garbage Collector führt in der Sweep-Phase eine Kompaktifizierung des Speichers durch, indem es alle erreichbaren Objekte an ein Ende des Heaps verschiebt.
- Beim Copying Garbage Collector Verfahren ist der gesamte Heap ist in zwei Hälften aufgeteilt.
  - Der Mutator arbeitet zunächst nur in einer Hälfte und allokiert dort Objekte, bis der freie Speicher in dieser Hälfte knapp wird.
  - Der Garbage Collector kopiert dann die erreichbaren Objekte in die andere Hälfte des Heaps.
  - Danach kann der Mutator weiter laufen wobei die Rollen der beiden Häften vertauscht sind.

Es dürfte klar sein, dass bei all diesen Verfahren das Anwendungsprogramm für eine längere Zeit unterbrochen wird.

Es gibt Verfahren, die die erste oder die zweite Phase des Garbage Collecting in kleinere Prozesse zerlegen, um die Pausen für das Anwendungsprogramm nicht zu lang werden zu lassen.
Da es hier aber zu Interaktionen mit dem Mutator kommen kann, müssen diese Verfahren beim Bestimmen der erreichbaren Objekte einen beträchtlichen Extra-Aufwand treiben, um zu verhindern, dass erreichbare Objekte fälschlicherweise dem freien Speicher zugeteilt werden.
Werden andrerseits nicht erreichbare Objekte nicht dem freien Speicher zugeteilt und geschieht dies bei nicht zu vielen Objekten, macht das Verfahren keine Probleme, da bei der nächten Aktivierung des Garbage Collectors diese Objekte garantiert nicht erreicht werden können.