TECHNISCHE UNIVERSITÄT
KAISERSLAUTERN

Software Entwicklung 1

Annette Bieniusa
Peter Zeller

AG Softech
FB Informatik
TU Kaiserslautern

Speichermanagement

Wie viel Speicher braucht ein Programm?

Wofür wird Speicher benötigt?

Wie ist der Speicher organisiert?

Wie wird der Speicher verwaltet?

Speicherverwendung

Wofür wird Speicher benötigt?

- Programmcode (unabhängig von Eingabedaten)
- Verwaltung von Prozedur-/Methodenaufrufen
- Werte, Objekte, Arrays, etc.

$\Rightarrow$ Abhängig von Laufzeitumgebung!

Speicherbedarf: Primitive Datentypen

In Java:

Datentyp	Bytes (1 Byte = 8 Bit)
boolean	1
char	2
int	4
float	4
long	8
double	8

In C: $\Rightarrow$ maschinenabhängig!

Speicherbedarf: Referenzen

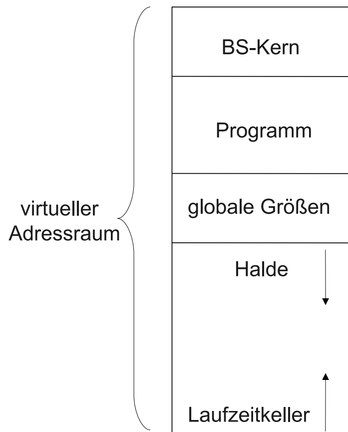
- Typischerweise abhängig von der konkreten Prozessorarchitektur
- In 32-Bit Architekturen: 4 Byte
- In 64-Bit Architekturen: 8 Byte

Speicherbedarf: Strukturen/Objekte und Arrays

- Bei Strukturen in C: abhängig von der Anzahl und Art der Komponenten
- Bei Objekten in Java: abhängig von der Anzahl und Art der Attribute + Overhead zur Objektverwaltung (Klasseninformation, GC, ...)
- Array mit N Elemente, wobei M Byte zum Speichern eines Elements benötigt werden: mind. $N \times M$ Byte zur Repräsentierung der Elemente (+ evtl. Overhead für Länge des Arrays, etc. in Java)

Wie ist der Speicher organisiert?

- **Programmbereich**: kompilierter Programmcode / Maschinenbefehle
- **Globalen Datenbereich**: alle Daten, die vom Beginn des Programms bis zum Programmende verfügbar sind (u.a. globale Variablen, String-Literale)
- **Stack** (dt. Laufzeitkeller): Prozedurinkarnationen mit ihren lokalen Variablen
- **Heap** (dt. Halde): dynamischen Speicherverwaltung



Speicherverwaltung

- **Allokation** (engl. *allocation*): Anfordern von Speicher
- **Deallokation** (engl. *deallocation*): Freigabe von Speicher
- Problem: **Speicherleck** (engl. memory leak)
  - Programm benötigt mit der Zeit immer mehr Speicher
  - Performance-Probleme oder auch Programmabbruch

→ Nicht mehr verwendeter Speicher sollte daher immer freigegeben werden!

Zwei Vorgehensweisen:

- Speicherbereinigung durch den Programmierer (Beispiel: C)
- Automatische Speicherbereinigung (Beispiel: Java)

Speicherbereinigung durch Programmierer (C)

- Speicherbereiche können auf Stack oder auf Heap reserviert werden
- Stack wird in C automatisch verwaltet
- Verwaltung des Heap durch `malloc`, `calloc`, `realloc`, `free`

+ sehr effizient und flexibel

- fehleranfällig (mehrfaches Freigeben, fehlende Initialisierung,...)
- schwierig zu debuggen

Automatische Speicherbereinigung

- Prozess ermittelt periodisch oder bei Bedarf, welcher Speicherbereich nicht mehr benötigt wird
- Automatische Deallokation (und evtl. Defragmentierung)

+ Vereinfacht die Programmierung
+ Kann effizient ausgeführt werden (z.B. in nebenläufigem Prozess)
- Benötigt zusätzlichen Rechenaufwand

TECHNISCHE UNIVERSITÄT KAISERSLAUTERN

Erreichbarkeit

Ein Objekt X heißt von einer Variablen v **direkt erreichbar**, wenn v eine Referenz auf X enthält.

X heißt von v **erreichbar**, wenn es von v direkt erreichbar ist oder wenn es ein Objekt Y mit Attribut w gibt, so dass X von w direkt erreichbar ist und Y von v erreichbar ist.

- Ermittele erreichbare Objekte ausgehend von **Wurzelvariablen**
- in Objekt heißt **erreichbar** *in einem Ausführungszustand* A, wenn es von einer Wurzelvariablen zu A erreichbar ist.
- Nicht-erreichbare Objekte können freigegeben werden

Verhindern von Speicherlecks bei GC

Aus der Implementierung der `java.util.ArrayList`:

```
public E remove(int index) {
    rangeCheck(index);

    modCount++;
    E oldValue = elementData(index);

    int numMoved = size - index - 1;
    if (numMoved > 0)
        System.arraycopy(elementData, index+1,
          elementData, index, numMoved);
    elementData[--size] = null;
          // clear to let GC do its work

    return oldValue;
}
```

Speicherlecks.... I

```
public static void main (String[] args) {
  int count = 1;
  while (true) {
    int[] ar = new int[1000000];
    StdOut.println(count++);
  }
}
```

- Kein Speicherleck!

Speicherlecks.... II

```
public static void main (String[] args) {
  List<int[]> la = new LinkedList<int[]>();
  int count = 0;
  while (true) {
    la.add(new int[1000000]);
    count++;
    StdOut.println(count);
  }
}
```

- Terminiert mit `OutOfMemoryError`