

Müllabfuhr (garbage collection)



Spezielle Literatur

jGuru: Fundamentals of RMI, Short Course.

<http://developer.java.sun.com/developer/onlineTraining/rmi/RMI.html>

Jones, R.; Lins, R.: Garbage Collection. John Wiley & Sons Ltd., 1996.

Plainfossé, D.; Shapiro, M.: A Survey of Distributed Garbage Collection Techniques. Int. Workshop on Memory Management, Kinross, Scotland (UK), Sept. 1995.

http://www-sor.inria.fr/publi/SDGC_iwmm95.html

Tel, G.; Mattern, F.: The Derivation of Distributed Termination Detection Algorithms from Garbage Collection Schemes. ACM TOPLAS 15:1, January 1993, pp. 1-35.

http://www.inf.ethz.ch/vs/publ/papers/distr_sim_dfg.pdf

Lokale Müllabfuhr

Referenzzähler



Idee:

- Jedes Objekt enthält einen Zähler, der angibt, wieviele Referenzen auf das Objekt verweisen.
- Objekte, deren Zähler den Wert 0 enthält, sind Müll.



Vorteile

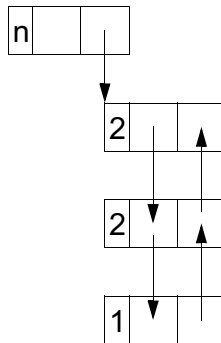
- Overhead wird über die gesamte Bearbeitung der Anwendung verteilt, also keine längeren Zeiten, in denen die Anwendung wegen der Müllabfuhr blockiert werden muß.
- Freigabe ist räumlich lokal, es müssen keine weiteren Teile des Adressraums modifiziert werden. Freigabe verträgt sich gut mit Demand Paging.
- Kurzlebige Objekte werden sehr schnell wieder entfernt.



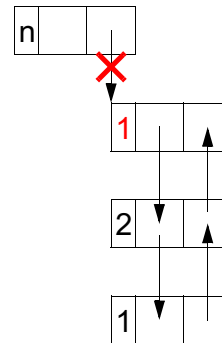
Nachteile

- Hoher Aufwand, um die Zähler auf dem Laufenden zu halten. Bei Modifikation einer Referenzvariablen müssen die Zähler des alten und des neuen Zielobjekts modifiziert werden.
- Jedes Objekt muß Platz für Zähler enthalten.
- Zyklische Datenstrukturen (z. B. doppelt verkettete Listen) müssen gesondert behandelt werden!

Wurzel



Wurzel



9.1.2 Mark and sweep



Idee

1. Phase: Alle erreichbaren Objekte markieren mit Hilfe einer Explosionswelle analog der Vorgehensweisen bei Wahlalgorithmen.
2. Phase: Freigabe aller nicht markierten Objekte. dabei eventuell Speicherkompression.



Vorteile

- Zyklen werden ganz natürlich mitbehandelt.
- Referenzmanipulation erfordert keinen Overhead.



Nachteile

- Während der Durchführung muß die Anwendung blockiert werden. Es existieren verschiedene Verfahren diesen Nachteil durch Partitionierung oder durch Nebenläufigkeit zu mindern.
- Mit zunehmender Nutzung des verfügbaren Speicherplatzes erhöhen sich Häufigkeiten und Dauer der Durchführung des "mark and sweep"-Algorithmus.
- Nur sehr bedingt echtzeitfähig.

9.2**Verteilte Müllabfuhr**

Deutlich schwieriger wegen eventueller Prozessor- oder Verbindungsausfälle, die unbedingt behandelt werden müssen!

Z. B. muss ein Server Objekte, die nur von einem ausgefallenen Rechner aus referenziert werden, freigeben können.

9.2.1**Präzisierung der Aufgabenstellung**

1. Das zu betrachtende System besteht aus einer Menge O von Objekten (Instanzen).
2. Es gibt eine Menge $R \subseteq O$ von Wurzelobjekten.
3. Objekte können andere referenzieren; eine Referenz zu einem Objekt r wird als r -Referenz bezeichnet.
4. Referenzen können als Parameter von Methodenaufrufen oder -ergebnissen übermittelt (gesendet) werden.
5. Objekt r ist direkt erreichbar von q genau dann, wenn q eine r -Referenz enthält oder ein Methodenaufruf an q mit einer r -Referenz als Parameter "unterwegs" ist.
6. Ein Objekt heißt erreichbar, wenn von einem Wurzelobjekt eine Folge direkter Erreichbarkeiten zu ihm führt.
7. Objekte die eine r -Referenz enthalten, können sie jederzeit tilgen.
8. Ein erreichbares Objekt kann eine r -Referenz, die es besitzt, kopieren und an andere Objekte senden.

- Semiformale Beschreibung der einschlägigen elementaren Aktionen
- ◆ Copy Reference
CR_p: { p ist erreichbar und hält eine r-Referenz }
 sende eine <copy r>-Nachricht an q;
- ◆ Receive Reference
RC_p: { Eine <copy r>-Nachricht ist angekommen }
 übernehme die <copy r>-Nachricht;
 füge die r-Referenz zu den vorhandenen Referenzen hinzu;
- ◆ Delete Reference
DR_p: { p besitzt eine r-Referenz }
 tilge die r-Referenz;

- Aufgabe der Müllabfuhr-Algorithmen ist es, nicht-erreichbare Objekte ausfindig zu machen und den von ihnen belegten Speicherplatz freizugeben. Sie müssen folgende Kriterien erfüllen
- ◆ G1 Sicherheit (Safety)
Es werden nur nicht-erreichbare Objekte als Müll eingesammelt.
- ◆ G2 Lebendigkeit (Liveness):
Wenn ein Objekt nicht erreichbar ist, wird es schließlich als Müll eingesammelt.

◆ Semiformale Beschreibung des Müllabfuhrproblems

CR_p : { p ist erreichbar und hält eine r-Referenz }

sende eine <copy r>-Nachricht an q;

RC_p : { Eine <copy r>-Nachricht ist angekommen }

übernehme die <copy r>-Nachricht;

füge die r-Referenz zu den vorhandenen Referenzen hinzu;

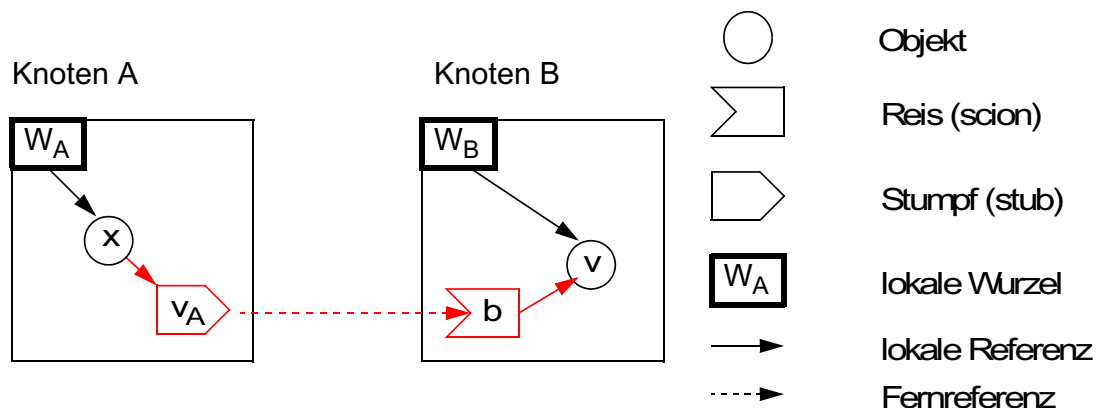
DR_p : { p besitzt eine r-Referenz }

tilge die r-Referenz;

9.2.2 Beispiel: Lokale (indirekte) Referenzenzählung

□ Die Problematik der Referenzenzählung in verteilten Systemen

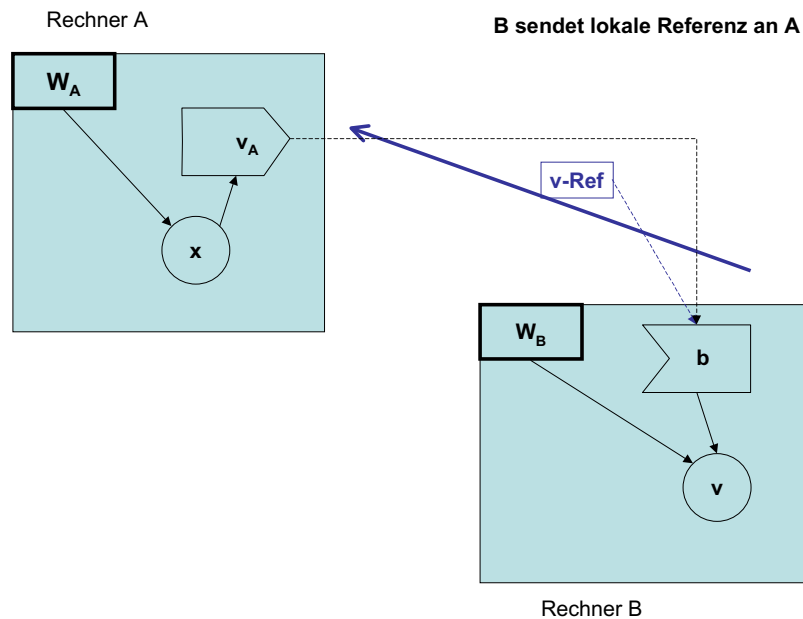
Darstellung von Fernreferenzen



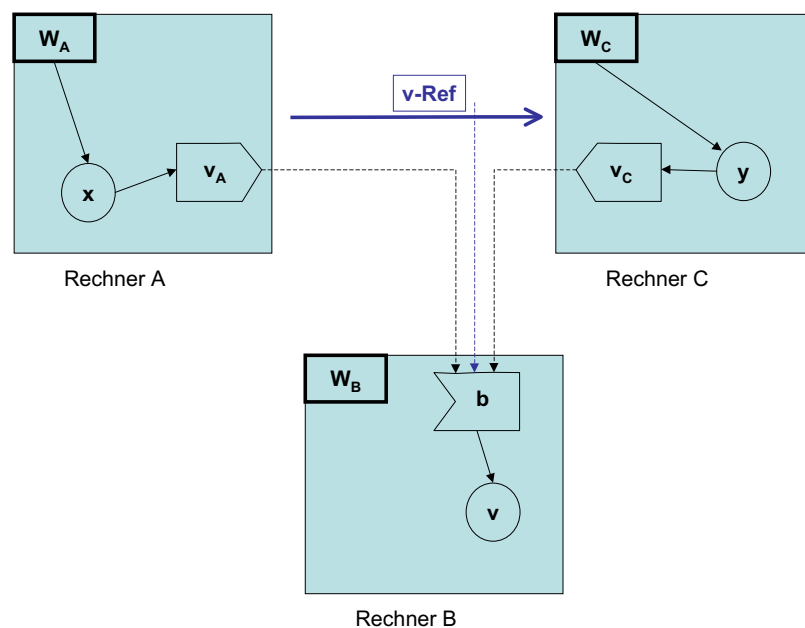


Die grundlegenden Aktionen

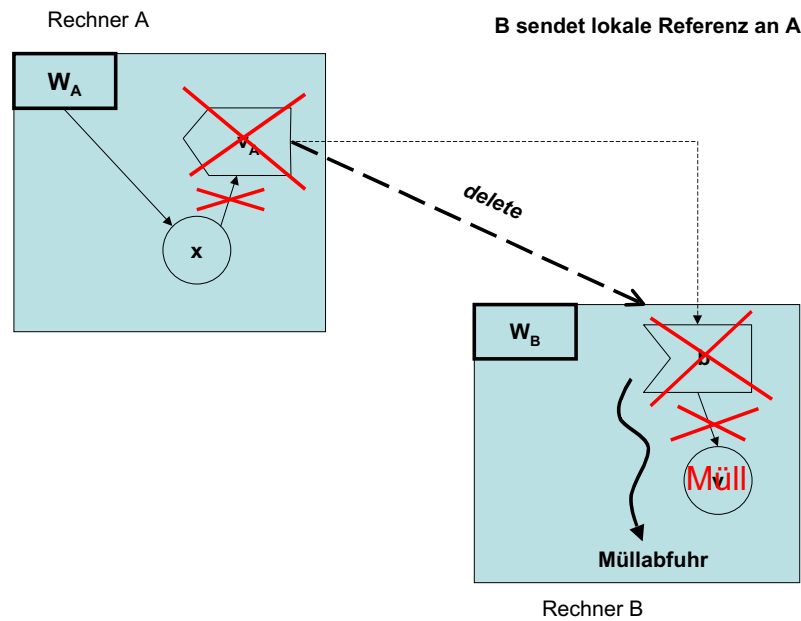
1. Erzeugung von Fernreferenzen



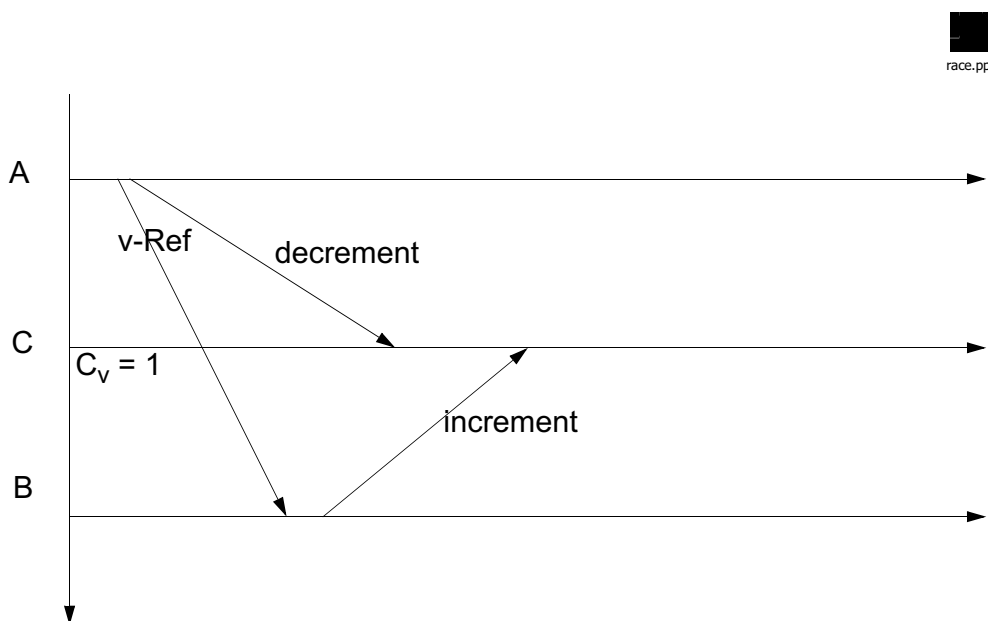
2. Verdopplung von Fernreferenzen



3. Tilgung von Fernreferenzen



Für Referenzzählung zusätzlich "increment"- und "decrement"-Nachrichten notwendig.
Einfache Realisierungen führen zu Wettlaufproblemen.



Mögliche Lösung: Lokale Referenzzählung (Ichisugi and Yonezawa, 1989)

Jedes Objekt o verfüge über Referenzzähler $LRC_p(o)$, die die von einem Objekt auf Prozessor p auf o verweisenden Referenzen zählen.

CR_p : { p besitzt eine o -Referenz }
 sende $\langle cop\ o \rangle$ an q ; $LRC_p(o)++$;

 RC_p : { Eine von q gesendete $\langle cop\ o \rangle$ -Nachricht hat p erreicht }
 empfangen $\langle cop\ o \rangle$; übernehme o -Referenz;
 if ($LRC_p(o) == 0$) { $LRC_p(o) = 1$; $FIRST_p(o) = q$; }
 else { sende $\langle dec\ o \rangle$ an q ; $LRC_p(o)++$; }

 DR_p : { p besitzt eine o -Referenz }
 tilge die o -Referenz; $LRC_p(o)--$;

 RD_p : { Eine $\langle dec\ o \rangle$ -Nachricht ist angekommen }
 empfangen $\langle dec\ o \rangle$; $LRC_p(o)--$;

 DZ_p : { $LRC_p(o)$ wurde gerade von 1 auf 0 reduziert }
 if ($FIRST_p(o) \neq \text{null}$;) sende $\langle dec\ o \rangle$ an $FIRST_p(o)$;
 else Speicherplatz von o freigeben

9.2.3 Bezug zum Terminierungsproblem

◆ Semiformale Beschreibung des Terminierungsproblems

S_p : { $state_p == \text{active}$ }
 sende eine $\langle M \rangle$ -Nachricht;

 R_p : { Eine Basisnachricht ist angekommen }
 empfangen Nachricht $\langle M \rangle$; $state_p = \text{active}$;

 I_p : { $state_p == \text{active}$ }
 $state_p = \text{passive}$

System wird ergänzt um ein Objekt A_p pro Prozeß p und ein Wurzelobjekt Z .

System wird so modifiziert, daß

1. A_p genau dann eine Z -Referenz besitzt, wenn p aktiv ist und
2. jede Basis-Nachricht als Parameter eine Z -Referenz mit sich trägt.

In dem so ergänzten System ist offensichtlich Terminierung äquivalent dazu, daß Z nicht mehr erreichbar ist.



Das führt zu folgender geänderten Übergangsbeschreibung:

- S_p : { $state_p == active$ }
 sende eine $\langle M, \text{<cop Z>}$ -Nachricht;
- R_p : { Eine Basisnachricht ist M angekommen }
 empfange Nachricht $\langle M, \text{<cop Z>}$; $state_p = active$;
 füge die Z-Referenz zu den vorhandenen Referenzen hinzu;
- I_p : { $state_p == active$ }
 $state_p = passive$;
 tilge alle Z-Referenzen;

9.2.4 Anwendung des Beispielalgorithmus zur Müllabfuhr auf dieses System

Offensichtlich muß zur Anwendung des Müllabfuhralgorithmus

- CR_p mit S_p kombiniert werden,
- RC_p mit R_p und
- DR_p mit I_p .

Außerdem ist es zweckmäßig, um nicht mehr als eine Z-Referenz pro Objekt führen zu müssen, eine zweite Z-Referenz durch einen zusätzlichen Übergang unmittelbar nach ihrer Entstehung wieder zu tilgen.

- (1) " A_p besitzt eine Z-Referenz" ist äquivalent zu " $state_p == active$ "
- (2) Da es nur Z-Referenzen gibt und jede Basisnachricht eine solche mit sich tragen muß, ist eine explizite Übertragung nicht erforderlich.

```
Sp:   { statep == active }
        sende eine Nachricht <M> an q; LRCp++;

Rp: { Eine Basisnachricht <M> von q ist angekommen }
        empfangen Nachricht <M>;
        if (LRCp == 0) {
            LRCp = 1; FIRSTp = q; statep = active;
        } else {
            sende <dec> an q;
            if (statep == passive) { statep = active; LRCp++; }
        }

Ip:   { statep == active }
        statep = passive;
        LRCp--;
        if (LRCp == 0) sende <dec> an FIRSTp;

RDp: { Eine <dec>-Nachricht ist angekommen }
        empfangen <dec>; LRCp--;
        if (LRCp == 0) sende <dec> an FIRSTp;
```

Anfänglich sei genau ein Prozeß aktiv und besitze eine Z-Referenz.
Entsprechend besitzt LRC für den Startprozeß S den Anfangswert 1, für alle anderen den Anfangswert 0.
Terminierung liegt vor, wenn $LRC_S == 0$ ist.