Script **generated by TTT**

Title: Grundlagen_Betriebssysteme (07.12.2012)

Date: Fri Dec 07 08:30:42 CET 2012

Duration: 79:30 min

Pages: 25

Adressräume

Maschinenadressraum

Arbeitsspeicher = Folge von fortlaufend nummerierten Bytes (ab 0). ***Maschinenadresse*** = Nummer des Bytes.

Programmadressraum

Programmadressen = Adressen im Programm (z.B. für Variable).

Die Menge der zulässigen Programmadressen ist der ***Programmadressraum*** . Dieser ist prozessspezifisch, d.h. Programmadressen haben nur Programm-lokale Bedeutung z.B. als Sprungziele.

Speicherabbildung

Abbildung der Programmadressen auf Maschinenadressen (durch CPU).

direkte Adressierung,

Basisadressierung,

Seitenadressierung und

Segment-Seitenadressierung.

Einführung

Die unmittelbare Nutzung des physischen Adressraums (Arbeitsspeichers) bei der Anwendungsentwicklung ist nicht empfehlenswert. Probleme sind folgende:

- Kenntnisse über Struktur und Zusammensetzung des Arbeitsspeichers notwendig.
- Kapazitätsengpässe bei der Arbeitsspeichergröße.

⇒ deshalb Programmerstellung unabhängig von realer Speichergröße und -eigenschaften.

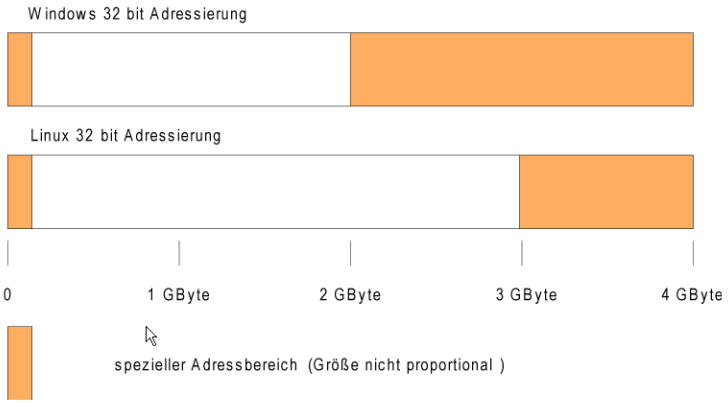
Single-Threaded Adressraum

Multi-Threaded Adressraum

Beispiel - Adressräume

32 Bit große Adressräume

Programmcode, statische Daten, Halde und Laufzeitkeller jeweils in einem für die Anwendung zugänglichen Bereich des Adressraums



Dynamisch erzeugte Daten, z.B. Listen und Objekte werden auf der Halde (heap) gespeichert. Die Halde wird nicht nach dem Kellerprinzip verwaltet.

Belegung Beispiel C:

`void malloc(size_t size)` belegt dynamisch einen Speicherbereich und gibt einen Pointer auf diesen Speicherbereich zurück.

Der Speicherbereich einer Halde besteht aus zwei Klassen:

Belegtbereiche : Speicherbereiche werden für Realisierungen von Datenobjekten verwendet.

Freibereiche : Speicherbereiche, die momentan nicht für Realisierungen von Datenobjekten verwendet werden; d.h. sie sind frei.

Buchführung des Speicherbereichs einer Halde mit Hilfe von Belegungs- und Freigabeoperationen.

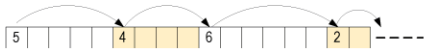
[Verwaltung der Freibereiche]
[Freiliste]
[Garbage Collection]
[Animation Haldenverwaltung]

Speicherbereiche (belegt, frei) sind in Blöcken unterschiedlicher Länge organisiert. Verwaltung freier Blöcke

implizite Liste über Längen - verlinkt alle Blöcke

jeweils Feststellen, ob der Block belegt oder frei ist.

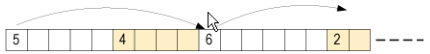
extra Bit im selben Wort wie Blockgröße.



Implementierung: sehr einfach; Allozierung in linearer Zeit.

explizite Liste (Freiliste) über die freien Blöcke

Allozierung: Durchsuchen der Freiliste nach geeignetem Block



Freiliste

Die Freibereiche der Halde werden mit Hilfe einer Liste (z.B. einfach verkettet) verwaltet.

```
public class freibereich {
    int size;
    freibereich next;
    <methodendefinitionen>
}
```

[Auswahl eines geeigneten Freibereichs]

Fragmentierung

Problematisch ist die Entstehung von vielen kleinen freien Rest-Freibereichen, die wegen ihrer kleinen Längen als Belegtbereiche ungeeignet sind.

Bei Erzeugung eines neuen Datenobjektes wird die Operation `belege(int size)` ausgeführt. Für die Auswahl eines geeigneten Freibereichs zur Erfüllung der gestellten Anforderungen existieren verschiedene Verfahren.

first-fit-Verfahren

bestimme von Beginn der Freiliste den ersten Speicherbereich, der die Anforderung erfüllt.

next-fit-Verfahren

bestimme in der Freiliste den ersten Speicherbereich, der die Anforderung erfüllt. Die Suche wird dort fortgesetzt, wo die letzte Suche beendet wurde.

best-fit-Verfahren

bestimmt in der gesamten Freiliste den Speicherbereich, der am besten die gestellte Anforderung erfüllt, d.h. mit möglichst wenig Verschnitt.

worst-fit-Verfahren

bestimme in der Freiliste den größten freien Speicherbereich und teile ihn in einen Belegtbereich (zur Erfüllung der Anforderung) und einen verbleibenden Freibereich auf.

Freiliste

Die Freibereiche der Halde werden mit Hilfe einer Liste (z.B. einfach verkettet) verwaltet.

```
public class freibereich {
    int size;
    freibereich next;
    <methodendefinitionen>
}
```

Auswahl eines geeigneten Freibereichs

Fragmentierung

Problematisch ist die Entstehung von vielen kleinen freien Rest-Freibereichen, die wegen ihrer kleinen Längen als Belegtbereiche ungeeignet sind.

Java sammelt belegten Speicher, der nicht mehr benötigt wird, auf.

automatische Rückgewinnung von nicht mehr benötigtem Speicher

üblich in Spachen wie Java, Lisp, Perl, ...

Woher kennt der Speichermanager nicht mehr benötigte Objekte?

i.a. nicht bekannt

allerdings können gewisse Blöcke nur dann benutzt werden, wenn Pointer zu Ihnen existieren.

Annahmen über Pointer

Speichermanager müssen Pointer von anderen Objekten unterscheiden können.

Alle Pointer müssen auf den Blockanfang zeigen.

Speicher als Graph

Zeitpunkt der Durchführung von Garbage Collection

wenn kein weiterer Platz mehr vorhanden ist, z.B. `new` gibt Fehler zurück

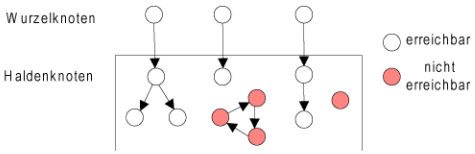
automatisch im Hintergrund, wenn System nicht ausgelastet ist.

Annahme

jeder Block ist ein Knoten

jeder Pointer ist eine Kante

Kantenausgangsknoten ist nicht auf der Halde (sondern im Keller) ⇒ Knoten ist Wurzel



Besttimmung erreichbarer Knoten.

Mark and Sweep

Aufsammeln mittels markieren und durchlaufen

Mark: an Wurzeln starten, Pfade durchlaufen und alle erreichbaren Speicherblöcke markieren

Sweep: Scannen aller Blöcke; Freigabe nicht markierter Blöcke

Java sammelt belegten Speicher, der nicht mehr benötigt wird, auf.

automatische Rückgewinnung von nicht mehr benötigtem Speicher

üblich in Spachen wie Java, Lisp, Perl, ...

Woher kennt der Speichermanager nicht mehr benötigte Objekte?

i.a. nicht bekannt

allerdings können gewisse Blöcke nur dann benutzt werden, wenn Pointer zu Ihnen existieren.

Annahmen über Pointer

Speichermanager müssen Pointer von anderen Objekten unterscheiden können.

Alle Pointer müssen auf den Blockanfang zeigen.

Speicher als Graph

Zeitpunkt der Durchführung von Garbage Collection

wenn kein weiterer Platz mehr vorhanden ist, z.B. `new` gibt Fehler zurück

automatisch im Hintergrund, wenn System nicht ausgelastet ist.

Dynamisch erzeugte Daten, z.B. Listen und Objekte werden auf der Halde (heap) gespeichert. Die Halde wird nicht nach dem Kellerprinzip verwaltet.

Belegung Beispiel C:

`void malloc(size_t size)` belegt dynamisch einen Speicherbereich und gibt einen Pointer auf diesen Speicherbereich zurück.

Der Speicherbereich einer Halde besteht aus zwei Klassen:

Belegtbereiche : Speicherbereiche werden für Realisierungen von Datenobjekten verwendet.

Freibereiche : Speicherbereiche, die momentan nicht für Realisierungen von Datenobjekten verwendet werden; d.h. sie sind frei.

Buchführung des Speicherbereichs einer Halde mit Hilfe von Belegungs- und Freigabeoperationen.

Verwaltung der Freibereiche

Freiliste

Garbage Collection

Animation Haldenverwaltung

Einführung

Die unmittelbare Nutzung des physischen Adressraums (Arbeitsspeichers) bei der Anwendungsentwicklung ist nicht empfehlenswert. Probleme sind folgende:

- Kenntnisse über Struktur und Zusammensetzung des Arbeitsspeichers notwendig.
- Kapazitätsengpässe bei der Arbeitsspeichergröße.

⇒ deshalb Programmerstellung unabhängig von realer Speichergröße und -eigenschaften.

Adressräume

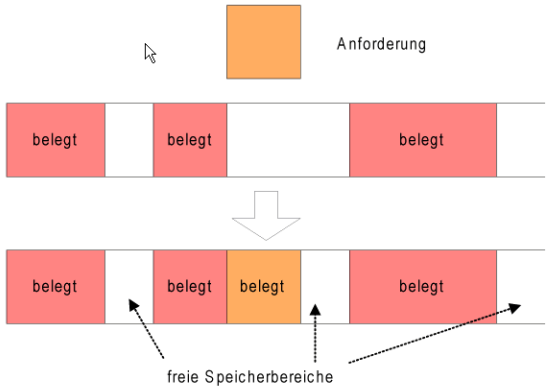
Organisation von Adressräumen

Fragmentierung

Forderungen an Adressraumrealisierung

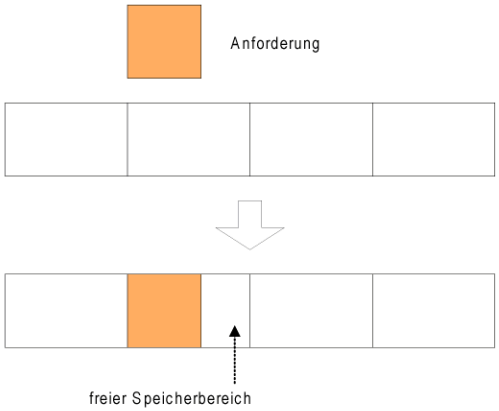
Externe Fragmentierung

Es wechseln sich benutzte und unbenutzte Speicherbereiche innerhalb des Adressraums ab.
Speicheranforderungen werden jeweils genau erfüllt.



Interne Fragmentierung

Der Speicher ist in Bereiche fester Größe untergliedert und Speicheranforderungen werden nur in Vielfachen dieser festen Grundgröße befriedigt.



Fragmentierung

Unter dem Begriff ***Fragmentierung*** versteht man verschiedene Formen der Zerstückelung des noch freien und nutzbaren Teils des Adressraums in kleine Bereiche. Unterscheidung zwischen ***externer*** und ***interner*** Fragmentierung.

Externe Fragmentierung

Interne Fragmentierung

Aus der Sicht der Anwendungsprogrammierung können für einen Adressraum eine Reihe wichtiger Forderungen an dessen Realisierung gestellt werden.

- Homogene und zusammenhängende Adressbereiche.
- Größe des genutzten Adressraums unabhängig von der Kapazität des physischen Adressraums (Arbeitsspeichers).
- Erkennen fehlerhafter Zugriffe.
- Erkennen von Überschneidungen zwischen Halde und Keller sowie zwischen mehreren Laufzeitkellern.
- Schutz funktionstüchtiger Anwendungen gegenüber fehlerhaften Anwendungen.
- Kontrollierbares und kontrolliertes Aufteilen der Speicherressourcen auf alle Anwendungen.
- Speicherökonomie, minimale Fragmentierung.

Speicherverwaltung

Fragestellungen

Dieser Abschnitt beschäftigt sich mit den Adressräumen für Programme und deren Abbildung auf den physischen Arbeitsspeicher einer Rechenanlage:

- Programmadressraum vs. Maschinenadressraum.
- Direkte Adressierung, Basisadressierung.
- Virtualisierung des Speichers; virtuelle Adressierung, insbesondere Seitenadressierung.

Einführung

Speicherabbildungen

Dieser Abschnitt behandelt einige Mechanismen zur Abbildung von Programmadressen auf Maschinenadressen des Arbeitsspeichers.

Direkte Adressierung

Basisadressierung

Seitenadressierung

Segment-Seitenadressierung

Speicherhierarchie / Caches

Programmadresse = Maschinenadresse; Probleme

Verschiebbarkeit,

Programmgröße,

Speicherausnutzung.

Verschiebbarkeit

Programmgröße

Programme größer als Arbeitsspeicher; Programmierer zerlegt Programm in Segmente (Nachladen bei Bedarf). Man spricht von der sogenannten ***Overlay-Technik*** (veraltet).

Forderung

Es muss daher gefordert werden, dass die Programmgröße unabhängig von der realen Arbeitsspeichergröße ist.

Speicherausnutzung

Programme bestehen aus Modulen, die nur zu bestimmten Zeiten verwendet werden.

Forderung

Arbeitsspeicher beinhaltet nur die momentan bzw. in naher Zukunft notwendigen Ausschnitte des Programms. Nutzen der ***Lokalitätseigenschaft*** von Programmen:

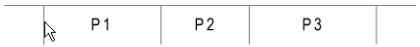
durch Datenstrukturen z.B. Arrays, oder

Programmstrukturen: Prozeduren, Schleifen.

In einem Mehrprozesssystem sind i.d.R. mehrere Programme im Arbeitsspeicher. Bei direkter Adressierung werden die Programme beim Laden fixiert und müssen dann bis zu ihrem Ende an derselben Stelle bleiben.

Problem

Externe Fragmentierung des Arbeitsspeichers. Sei der Arbeitsspeicher zunächst lückenlos mit Programmen P1, P2, P3 gefüllt.



Nach Beendigung des Programms P2 entsteht eine Lücke.

Forderung

In Mehrprogramme/Mehrprozesssystemen müssen daher Programme ***verschiebbar*** sein.

Direkte Adressierung

Programmadresse = Maschinenadresse; Probleme

Verschiebbarkeit,

Programmgröße,

Speicherausnutzung.

Verschiebbarkeit

Programmgröße

Programme größer als Arbeitsspeicher; Programmierer zerlegt Programm in Segmente (Nachladen bei Bedarf). Man spricht von der sogenannten ***Overlay-Technik*** (veraltet).

Forderung

Es muss daher gefordert werden, dass die Programmgröße unabhängig von der realen Arbeitsspeichergröße ist.

Speicherausnutzung

Programme bestehen aus Modulen, die nur zu bestimmten Zeiten verwendet werden.

Forderung

Arbeitsspeicher beinhaltet nur die momentan bzw. in naher Zukunft notwendigen Ausschnitte des Programms. Nutzen der ***Lokalitätseigenschaft*** von Programmen:

durch Datenstrukturen z.B. Arrays, oder

Programmstrukturen: Prozeduren, Schleifen.

Generated by Targeteam

Die Basisadressierung hat eine einfache Abbildungsvorschrift:

Maschinenadresse = Basisadresse + Programmadresse

Die Basisadresse ist programmspezifisch.

Die Programmadressen aller Programme beginnen jeweils mit Null.

Speicherverwaltungsstrategien

Aufgabe: Finden eines zusammenhängenden Arbeitsspeicherbereichs, der groß genug ist, um das Programm zu speichern ⇒ siehe Verfahren zur Verwaltung der Halde .

first-fit

Durchsuche die Liste der Freibereiche vom Anfang an und nimm den ersten passenden Frei-Bereich: Spalte nicht benötigten Speicher ab und füge ihn als freien Bereich in die Freibereichsliste ein.

next-fit

Durchsuche die Liste der Freibereiche nach first-fit, jedoch beginne die Suche dort, wo man bei der letzten Suche aufgehört hat.

best-fit

Durchsuche die Liste der Freibereiche vom Anfang an und nimm den passenden Frei-Bereich, der die Speicheranforderungen des Programms am besten erfüllt: Spalte nicht benötigten Speicher ab und füge ihn als freien Bereich in die Freibereichsliste ein.

worst-fit

Durchsuche die Liste der Freibereiche vom Anfang an und nimm den Frei-Bereich, der die Speicheranforderungen des Programms am schlechtesten erfüllt: Spalte nicht benötigten Speicher ab und füge ihn als freien Bereich in die Freibereichsliste ein.

Buddy-Systeme

Buddy-Systeme

Speicheranforderungen werden in Größen von Zweierpotenzen vergeben, d.h. eine Anforderung wird auf die nächste Zweierpotenz aufgerundet

Anforderung 280 Bytes ⇒ Belegung von 512 Bytes = 2^9

am Anfang besteht der Arbeitsspeicher aus einem großen Stück.

Speicheranforderung von 2^k : ist Speicherbereich dieser Größe vorhanden, dann Unterteilung eines Speicherbereichs der Größe 2^{k+1} in 2 Speicherbereiche der Größe 2^k Bytes.

Freigabe eines Speicherbereichs von 2^k : falls auch entsprechendes Partnerstück frei ist, dann Verschmelzung der beiden Speicherbereiche zu einem Speicherstück der Größe 2^{k+1} .

Fragestellungen

Dieser Abschnitt beschäftigt sich mit den Adressräumen für Programme und deren Abbildung auf den physischen Arbeitsspeicher einer Rechenanlage:

Programmadressraum vs. Maschinenadressraum.

Direkte Adressierung, Basisadressierung.

Virtualisierung des Speichers; virtuelle Adressierung, insbesondere Seitenadressierung.

Speicherabbildungen

Dieser Abschnitt behandelt einige Mechanismen zur Abbildung von Programmadressen auf Maschinenadressen des Arbeitsspeichers.