1 HINTERGRÜNDE/ MOTIVATION 2

2 FUNKTIONEN DER SPEICHERVERWALTUNG 2

3 ARTEN DER SPEICHERVERWALTUNG 2

3.1 STATISCHE SPEICHERVERWALTUNG 2
3.2 DYNAMISCHE SPEICHERVERWALTUNG 3
3.2.1 REALER SPEICHER OHNE AUSLAGERUNG 3
3.2.2 REALER SPEICHER MIT AUSLAGERUNG 3
3.2.3 VIRTUELLE SPEICHERVERWALTUNG 3
3.2.3.1 ZIELE 3
3.2.3.2 VERWIRKLICHUNG 3
3.2.3.2.1 SEGMENTIERUNG 5
3.2.3.2.2 SEITENVERWALTUNG 5

4 PROBLEME DER VIRTUELLEN SPEICHERVERWALTUNG 6

4.1 PROBLEME BEI DER SEGMENTVERWALTUNG 6
4.2 PROBLEME BEI DER SEITENVERWALTUNG 6
4.2.1 ZUSAMMENHANG ZWISCHEN FRAGMENTIERUNG UND SEITENGRÖßE 7

5 SCHUTZMECHANISMEN 7

1 Hintergründe/ Motivation

Die Speicherverwaltung gehört zu den Komponenten eines Betriebssystems. Dieses verwaltet den Speicherplatz als Betriebsmittel.
Ein aktiver Prozeß kann nur auf Elemente zugreifen, die sich im Hauptspeicher befinden. Der Hauptspeicher wiederum hat eine begrenzte Kapazität. Als Primärspeicher sollen trotz wahlfreiem Zugriff die Zugriffszeiten möglichst gering bleiben. Eine höhere Kapazität würde längere Zugriffszeiten bedingen, niedrigere Zugriffszeiten würden höhere Stückkosten verursachen. Ziel ist es also ,ein Gleichgewicht zwischen Stückkosten und Kapazität zu suchen.
Die heutige Rechnergeneration verwendet mehrstufige, hierarchische Speichersysteme. Es gibt einmal den primären Speicherbereich. Dem zuzuordnen sind die Register mit Zugriffszeiten gleich der CPU-Taktung, der First-Level-Cache mit halber CPU Taktung und Speichergrößen von 32-64 KB , der Second-Level-Cache mit Zugriffszeiten von 2,5 ns und einer Größe bis maximal 2 MB und der Hauptspeicher mit Zugriffszeiten von 6-7 ns und einer maximalen Größe von 4 GB (üblich heute 64 –128 MB) .
Den sekundären Speicherbereich bilden die Festplatte und Disketten, CD's oder Zip-Laufwerke.
Die heutigen Festplatten haben gewöhnlich eine Größen von 2 bis hin zu 40 GB und Zugriffszeiten von 5 bis 10 ms.
An dem Gefälle der Zugriffszeiten erkennt man deutlich die Notwendigkeit einer effizienten Hauptspeicherverwaltung, um eine schnelle Abarbeitung der Prozesse und eine optimale Ausnutzung der CPU überhaupt gewährleisten zu können.

2 Funktionen der Speicherverwaltung

Man unterscheidet bei der Speicherverwaltung zwischen Adreß- und Speicherraum. Unter dem Adreßraum versteht man alle in einem Speicher oder in einer Speicherhierarchie angeordneten Objekte , d.h. also alle adressierbaren Elemente.
Der Speicherraum bezeichnet nur die im Hauptspeicher befindlichen Elemente.
Dies bedeutet, daß die Größen von Adreß- und Speicherraum sehr stark differieren können. Die Speicherverwaltung muß also eine Beziehung zwischen Adreß- und Speicherraum herstellen.

3 Arten der Speicherverwaltung

3.1 Statische Speicherverwaltung

Statische Speicherverwaltung bedeutet, daß der Speicher nur für ein festes Zeitintervall vergeben werden kann. Es muß also vorher schon bekannt sein, welches Programm wieviel Speicher benötigt. Und auch die Speicheradressen werden zum Zeitpunkt des Compilierens schon festgelegt.

Dieses System ist durch das Betriebssystem einfach zu handhaben , jedoch auch relativ unflexibel. Der Speicherplatz ist begrenzt und andere Programme werden in ihrer Ausführung behindert.

3.2 Dynamische Speicherverwaltung

3.2.1 Realer Speicher ohne Auslagerung

Bei der realen Speicherverwaltung wird nur der physikalisch vorhandene Hauptspeicherplatz ausgenutzt. Ganze Programme werden also eingelagert, abgearbeitet und dann erst wieder ausgelagert. Daraus resultiert eine limitierte Größe von dem zur Verfügung stehenden Hauptspeicherplatz. Ein Beispiel für den Einprogrammbetrieb ist das Betriebssystem MS-DOS. Es unterstützt einen Adreßraum von 1 MB. Selbst belegt es jedoch schon einen Bereich von ca. 400 KB, so daß ein Anwenderprogramm die Größe von 600 KB nicht überschreiten darf.

3.2.2 Realer Speicher mit Auslagerung

Eine Verbesserung dieser Methode wäre eine ständige Ein- bzw. Auslagerung aktiver bzw. gesperrter Prozesse, um so zwischenzeitlich das Betriebsmittel Hauptspeicher wieder freigeben zu können. Zum Beispiel könnte ein Programm, welches vielleicht auf eine Eingabe wartet und deswegen gesperrt ist, zwischenzeitlich ausgelagert werden und ein neues Programm eingelagert werden , das den Status “bereit” hat. Eine weitere Verbesserung wäre die Unterteilung von Programmen in logische Tasks, so daß mehrere Tasks zur gleichen Zeit im Hauptspeicher gehalten werden. Dies erforderte eine relative Speicheradressierung , welche Aufgabe des Compilers ist.

3.2.3 Virtuelle Speicherverwaltung

3.2.3.1 Ziele

1. Jeder Prozeß sollte über einen Adreßraum verfügen, der so groß ist wie die gesamte Adressierungsbreite des Systems.
2. Der reale Hauptspeicher soll von möglichst vielen Anwendern gleichzeitig benutzt werden können.
3. Die Ausnutzung des Realspeichers soll möglichst effizient sein.

3.2.3.2 Verwirklichung

Die virtuelle Speicherverwaltung fordert eine Unterteilung des Programms in kleine Verwaltungseinheiten, im einfachsten Fall eine Unterteilung in Code- und Datenfelder. Die einzelnen Programmsegmente müssen nicht alle permanent und zur gleichen Zeit im Hauptspeicher gehalten werden. Nicht benötigte Teile werden auf den Hintergrund-

speicher ausgelagert. Somit erhöht sich die Anzahl aktiver Prozesse im Hauptspeicher.

Die virtuelle Speicherverwaltung hat somit höhere Anforderungen an das Betriebsystem. Es sind verschiedene Aufgaben zu bewältigen:

1 Der Einlagerungszeitpunkt

Als Einlagerungszeitpunkt bezeichnet man den Zeitpunkt, zu dem eine Anforderung an eine Verwaltungseinheit anliegt, die sich aber zur Zeit nicht im Hauptspeicher befindet. Ausgelöst wird die Einlagerung durch eine interne Unterbrechung (Seitenfehler). Daraufhin wird das entsprechende Segment von der Festplatte in den Hauptspeicher geladen.

2 Das Zuweisungsproblem

Soll ein neues Segment in den Hauptspeicher geladen werden, gilt es festzulegen, an welche Stelle des Hauptspeichers das Segment geladen werden soll. Es muß ein von der Größe des zu ladenden Segmentes abhängiger, zusammenhängender Speicherplatz gefunden werden. Bei unterschiedlichen Segmentgrößen, die ein- bzw. ausgelagert werden, entstehen unterschiedlich große Lücken und es ist nun Aufgabe des Betriebssystems eine geeignete Lücke zu finden.
Für diese Aufgabe gibt es drei verschiedene Lösungsansätze:

- first-fit-Methode
  Das Segment wird in die erste passende Lücke geladen
- best-fit-Methode
  Das Segment wird in die kleinstmögliche Lücke geladen
- worst-fit-Methode
  Das Segment wird in die größtmögliche Lücke geladen
  (Vermeidung kleiner, unbrauchbarer Lücken)

3 Das Ersetzungsproblem

Falls zum Moment des Einlagerungszeitpunktes keine freien Lücken im Hauptspeicher vorhanden sind, müssen vorhandene Segmente ausgelagert werden. Für die Ersetzung der im Hauptspeicher befindlichen Segmente gibt es ebenfalls drei Lösungsansätze:

- *FIFO-Prinzip (first in first out)*
  Es wird das Segment ersetzt, welches sich am längsten im Hauptspeicher befindet.
- *LRU-Prinzip (least recently used)*
  Auslagerung des Segmentes, auf welches am längsten schon nicht mehr zugegriffen wurde
- *LFU (least frequently used)*
  Das Segment mit der geringsten Anzahl von Zugriffen wird ausgelagert.

In der Praxis wird meist das LRU-Prinzip verwirklicht, welches durch Setzen eines Zugriffsbit realisiert wird. Wird auf ein Segment zugegriffen , so wird das Bit gesetzt. In bestimmten Zeitintervallen werden diese Zugriffsbits wieder zurückgesetzt, so daß man über das Bit ermitteln kann, ob auf das Segment im zurückliegenden Zeitintervall zugegriffen wurde.

3.2.3.2.1 Segmentierung

Die Segmentierung ist eine Form der virtuellen Speicherverwaltung. Bei der Segmentierung wird der Adreßraum in Blöcke variabler Größe unterteilt, die sogenannten Segmente, die der logischen Partitionierung des Programms entsprechen.
Die Adressierung dieser Segmente erfolgt über relative Adressen , die durch ein Wertepaar (u,v) beschrieben werden. U enthält den Segmentnamen und v den Verschiebeanteil (offset) , das heißt also die Position des Objektes innerhalb des Segmentes.
Für jeden Prozeß wird eine Segmenttabelle angelegt, in welche für jedes Segment ein Eintrag gemacht wird, der z.B. die Startadresse enthält, die Länge des Segmentes und auch eventuelle Zugriffsrechte. Ein Hardware-Register enthält die Zuordnungen zwischen Prozessen und Segmenttabellen. Somit kann die Adreßrechnung wie folgt vollzogen werden:
Als erstes wird die entsprechende Segmenttabelle geladen, für das u-te Element wird daraus die Startadresse ermittelt, von dort aus wird direkt auf das gewünschte Element zugegriffen, dessen Position über v (den offset) definiert ist.
Zusätzlich kann kontrolliert werden, ob man auf zugeteilten Speicherbereich zugreift. Die in der Segmenttabelle abgelegte Länge des Segmentes muß größer als der offset sein. Ansonsten wird ein Segmentfehler identifiziert , der einen internen Interrupt auslöst.

3.2.3.2.2 Seitenverwaltung

Die Seitenverwaltung ist eine weitere Form des virtuellen Speicherkonzeptes. Ihr wesentliches Merkmal ist die feste Größe der Programmblöcke.
Der verfügbare Adreßraum wird in logische Blöcke gleicher Größe unterteilt , die Seiten(oder auch pages), und der Speicherraum wird in physikalische Blöcke gleicher Größe aufgeteilt, die Seitenrahmen.
Adressierung von Elementen erfolgt auch wieder relativ über das Wertepaar (u,v) , wobei u die Seitennummer und v den offset enthält.
Auch hier gibt es Seitentabellen und Seitentabellen-Register, ähnlich der Segmentverwaltung eben nur ohne Längenangaben, da diese ja konstant sind. Typische Seitengrößen sind 512 Bytes, 1K, 2K, 4K und so weiter.

4 Probleme der virtuellen Speicherverwaltung

Die virtuelle Speicherverwaltung verursacht einen hohen Verwaltungsaufwand. Für die Effizienz dieses Konzeptes muß also gewährleistet sein, daß der Verwaltungsaufwand nicht zu hoch wird.
Wird nämlich die Anzahl aktiver Prozesse bezogen auf den realen Hauptspeicherplatz zu groß, so steigt auch sehr schnell die Anzahl der ein- bzw. auszulagernden Teile pro Zeiteinheit, man spricht auch vom exponentiellen Anstieg der Pagingrate. Durch diesen Zustand, auch Trashing genannt, wächst der Verwaltungsaufwand unverhältnismäßig.
Erhält ein Prozeß Prozessorzeit, müssen zuerst notwendige Bereiche eingelagert werden. Danach wird schon der nächste Prozeß aktiviert, und es besteht die Gefahr, daß die gerade eingelagerten Seiten wieder ausgelagert werden müssen.
Außerdem sollte auch eine nahezu maximale Ausnutzung des Speicherraumes erreicht werden.

4.1 Probleme bei der Segmentverwaltung

Da hier die Fragmente unterschiedlich groß sind , können “Löcher” entstehen, die dann ungenützt bleiben.
Man spricht auch von externer Fragmentierung. Der Speicherplatz wird dann nicht mehr optimal ausgenutzt. Zur Vermeidung von ungenutztem Speicherplatz über einen längeren Zeitraum ist dann eine Kompaktifizierung in regelmäßigen Zeitabständen notwendig. Auch dies wird vom Betriebssystem gesteuert, welches über Kontrollprogrammen feststellt, ob eine Kompaktifizierung nötig ist oder nicht.

4.2 Probleme bei der Seitenverwaltung

Das Seitenverfahren löst zwar das Fragmentierungsproblem innerhalb des Speichers , jedoch kann es hier sein, daß der Platz innerhalb einer Seite nicht völlig ausgenutzt wird. So kann es zum Beispiel sein, daß die letzte Seite eines Programms die Verwaltungseinheit nicht vollständig ausnützt , so daß kleinere Speicherbereiche brach liegen, dies jedoch nur für die Verweildauer einer Seite im Speicher. Man spricht auch von interner Fragmentierung.

4.2.1 Zusammenhang zwischen Fragmentierung und Seitengröße

Die Ausnutzung des Speichers hängt nicht nur von den Ersetzungs– und Ladestrategien ab, sondern auch von der Größe der Speicherseite. Von ihr hängt unmittelbar ab

a) die interne Fragmentierung: je größer die Seite desto größer der unausgenutzte Speicherplatz innerhalb der letzten Seite eines Programms
b) die Frequenz des Seitenwechsels: je kleiner die Seite, desto häufiger muß ein Seitenwechsel vollzogen werden

Dies führt zu folgendem Konflikt: Wählt man möglichst große Seiten, so erreicht man einen niedrigeren Verwaltungsaufwand durch die geringere Anzahl der Seitenwechsel, jedoch erhöht sich auch die interne Fragmentierung. Wählt man aber kleinere Seiten, so sinkt die interne Fragmentierung, aber es steigt der Verwaltungsaufwand für die häufigen Seitenwechsel.
Ziel ist also eine optimale Seitengröße, bei der möglichst beides, d.h. die interne Fragmentierung und die Anzahl der Seitenwechsel , minimiert wird.
Dazu können gewisse Berechnungen durchgeführt werden, anhand welcher dann die Größe einer Seite festgelegt werden kann. Die Seitengrößen differieren jedoch von Betriebssystem zu Betriebssystem, z.B. ist der Standard bei Windows NT 4 KB oder bei UNIX 512 Byte.

5 Schutzmechanismen

Sind mehrere Prozesse gleichzeitig aktiv, so müssen gewisse Schutzvorkehrungen getroffen werden. Ein Prozeß muß vor den Zugriffen anderer Prozesse geschützt werden, damit diese seine Daten und seinen Code nicht zerstören. Verhindert wird dieses eigentlich schon durch die relative Adressierung und Prüfen der Bereichsüberschreitung (wie oben bereits erwähnt) , durch welche dann ein Seitenfehler ausgelöst wird.
Jedoch muß es aber auch möglich sein, fremden Code oder fremde Datenbereiche zu nutzen. Zum Beispiel ist es teilweise erforderlich, daß Benutzerprozesse auf Betriebssystem-Code zugreifen (z.B. Drucker-Spooler) und umgekehrt müssen auch Systemroutinen auf Datenbereiche des Benutzers zugreifen können, z.B. der Editor. Dies wird dann über die Vergabe von Zugriffsrechten geleistet.
Zudem muß aber auch gewährleistet werden, daß ein Prozeß nicht durch seine eigenen Aktivitäten, wie z.B. durch Überschreiben seines eigenen Programmcodes, gestört wird. Lösung für dieses Problem kann die strikte Trennung von Programmcode und Daten sein, wenn man dann Programmcode als unveränderbar kennzeichnet.
dabei entfällt dann auch das Wegschreiben der Codeseiten oder – segmente auf den Hintergrundspeicher.