I. VIRTUELLE SPEICHERVERWALTUNG

Notwendigkeit

Immer größere Programme

Immer mehr Programme „gleichzeitig“

⇝ verfügbarer Arbeitsspeicher schnell zu klein

Lösung: Nur gerade benötigte Teile der aktiven Programme im Arbeitsspeicher, Rest bei Bedarf aus Hintergrundspeicher nachladen (*swapping*, *paging*)

Umsetzung: **MMU** (*memory management unit*) setzt virtuelle Adressen in physikalische um

Virtueller Speicher

Speicherkapazität größer als effektive Hauptspeicherkapazität

Betriebssystem lagert nach Bedarf Speicherbereiche ein/aus

MMU-Adressberechnung hardwaremäßig eindeutig

Abbildungsinformation in Übersetzungstabellen gespeichert

⇝ Abbildungsinformation für zusammenhängende Adressbereiche, um Übersetzungstabellen klein zu halten

Virtueller Speicher – Verwaltung (Segmentierung)

Virtueller Adressraum wird in Segmente verschiedener Länge zerteilt

Mehrere Segmente pro Programm (zB für Programmcode, Daten)

Segmente enthalten logisch zusammenhängende Informationen, relativ groß

Vorteile:
- spiegelt logische Programmstruktur wieder
- große Segmente ⇝ relativ seltener Datentransfer

Nachteile:
- Datentransfer umfangreich falls notwendig
- Programm aus nur einem Code- und Datensegment
  ⇝ muss vollständig eingelagert werden

Virtueller Speicher – Verwaltung (Seiten)

logischer und physikalischer Adressraum in Teile fester länge (Pages) zerteilt

Pages relativ klein (256-4k Byte)

Viele Seiten pro Prozess, keine logischen Zusammenhänge

Vorteile:
- kleine Seiten ⇝ nur wirklich benötigter Programmteil wird eingelagert
- geringerer Verwaltungsaufwand als Segmentierung

Nachteile:
- häufigerer Datentransfer als bei Segmentierung

Segmentbasierte Speicherverwaltung – Implementierung

virtuelle Adresse wird in **Segmentnummer** (n höherwertigste Bits der virtuellen Adresse) als Segmentkennung und in **Bytenummer** (verbleibende m Bits der virtuellen Adresse) als Abstand zum Segmentanfang unterteilt

max. virtuelle Segmentanzahl 2^n, max. Segmentgröße 2^m

Adressabbilung über Segmenttabelle (im MMU-Registerspeicher)

reale Adresse = Segmentbasisadresse + virtuelle Byte-Nummer

Segmentlängenangaben um segmentüberschreitende Zugriffe feststellen/verhindern zu können

Verschnitt: ungenutzter Raum bei Segmenten kleiner als 2^m

⇝ gute Hauptspeicherausnutzung, wenn Segmentgrenzen an jeder Byteadresse zulässig sind

Realität: Segmentgrenzen an Vielfachen von Blöcken (hier 256 Bytes)

Segmente im virtuellen physikalischen Adressraum in 256-Byte-Blöcke unterteilt

⇝ m-Bit-Bytenummer wird aufgeteilt in kürzere Bytenummer für Byteadressierung in Block und Blocknummer

Adressumsetzung: virtuelle Segmentnummer wird auf reale 24-Bit-Blocknummer als Segmentbasis abgebildet

Virtuelle Bytenummer für Adressierung innerhalb des Blocks wird unverändert übernommen

Virtuelle Speicherverwaltung – Probleme

Einlagerungszeitpunkt: Wann werden Segmente/Seiten in Hauptspeicher eingelagert?

Zuweisungsproblem: Wo in Hauptspeicher werden Seiten/Segmente eingelagert?

Ersetzungsproblem: Welche Segmente/Seiten auslagern um Platz für neu benötigte Daten zu schaffen?

Probleme – Einlagerungszeitpunkt

Gängiges Verwahren: Einlagerung auf Anforderung (*demand paging* bei Seitenverfahren)

Daten werden eingelagert, sobald auf sie zugegriffen wird, aber nicht in Hauptspeicher liegen

Segment-/Seitenfehler (*segment/page fault*): Zugriff auf nicht in Hauptspeicher vorhandene(s) Segment/Seite

Probleme – Zuweisungsproblem (Segmentierung)

ausreichend große Lücke in Hauptspeicher muss gefunden werden

Strategien:

1. **first-fit**: erste passende Lücke wird genommen
2. **best-fit**: kleinste passende Lücke wird genommen
3. **worst-fit**: größte passende Lücke wird genommen

Problem: Speicher zerfällt in belegte und unbelegte Speicherbereiche
⇝ **externe Fragmentierung**

unbelegte Speicherbereiche oft zu klein um weitere Segmente aufnehmen zu können

Probleme – Zuweisungsproblem (Seiten)

Problem taucht nicht auf, da alle Seiten gleich groß
⇝ es entstehen immer passende Lücken
⇝ **keine externe Fragmentierung**

Interne Fragmentierung: Einheitliche Seitengröße ⇝ auf letzter Seite des Programm-Moduls wahrscheinlich ungenutzter Leerraum

Probleme – Ersetzungsproblem (Segmentierung)

Limitierung der Anzahl gleichzeitig von einem Prozess benutzbaren Segmente

⇝ bei Einlagerung eines neuen Segments wird anderes Segment des Prozesses ausgelagert

Probleme – Ersetzungsproblem (Seiten)

Seitenersetzungsstrategien:

1. **FIFO** (*first in first out*): älteste Seite wird ersetzt
2. **LIFO** (*last in first out*): jüngste Seite wird ersetzt
3. **LRU** (*last recently used*): am längsten unbenutzte Seite wird ersetzt
4. **LFU** (*least frequently used*): am seltensten benutzte Seite wird ersetzt
5. **LRD** (*least reference density*): Seite mit geringster Zugriffsdichte wird ersetzt