Hasso Plattner Institut

IT Systems Engineering | Universität Potsdam

Speicherverwaltung, Interrupts & Exceptions

Daniel Richter

Seminar „Prozesssteuerung und Robotik“

03. Dezember 2008

Agenda

- Speicherverwaltung
  - Grundlagen
  - Möglichkeiten auf verschiedenen Ebenen
    - CPU/Hardware, Betriebssystem, virtuelle Maschinen
  - Algorithmen zur Speicherbereitstellung
- Interrupts & Exceptions
  - Interruptbehandlung
  - Maskierung, Prioritäten
  - Exceptions

Hasso Plattner Institut

IT Systems Engineering | Universität Potsdam

Speicherverwaltung

Motivation

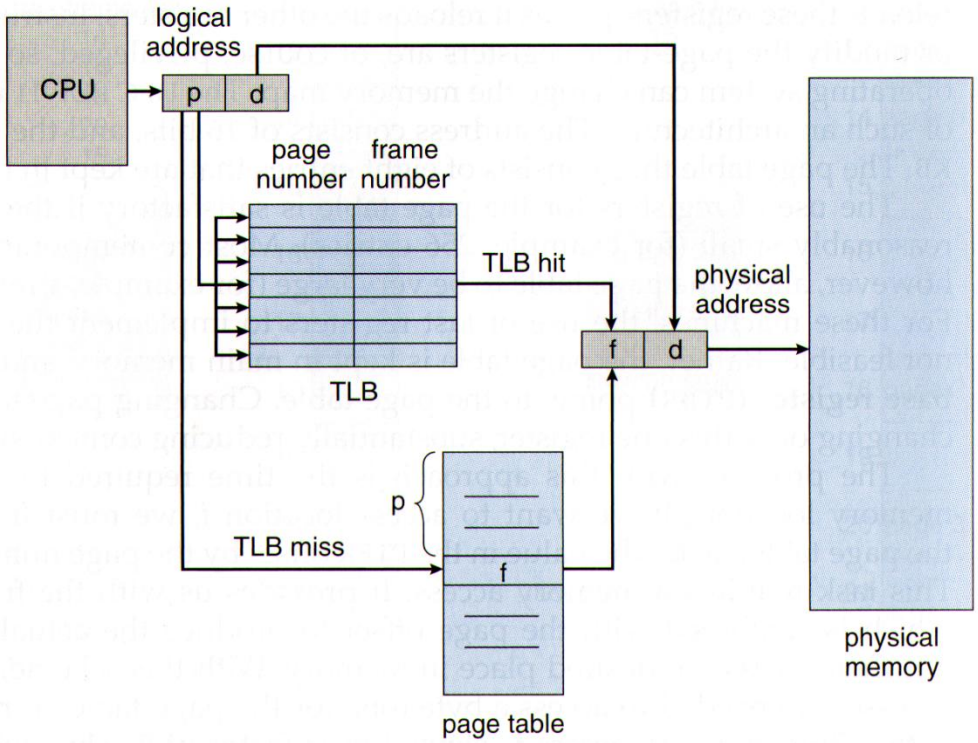
- verschiedenen Prozesse/Tasks mit eigenem Speicher
- Aufgaben
  - Speicher**bereitstellung** – Applikationen den Speicher geben, den sie brauchen
  - Speicher**abbildung** – Adressen, die im Prozess verwendet werden, auf echten Speicher abbilden
  - Speicher**schutz** – Verhindern, dass auf Speicher zugegriffen wird, der nicht zugeteilt war
- Herausforderung für eingebettete Systeme: Wie erhält man **vorhersagbares** Verhalten?

Grundlagen

- Segmentierung
  - Aufteilung des Speichers
  - Segmenttabelle (Name/Adresse, Länge/Offset, evtl. unterschiedliche Attribute)
  - statt großem, zusammenhängendem Bereich mehrere kleine
  - Problem: interne & externe Fragmentierung
- Paging/virtueller Speicher
  - virtuelle vs. logische vs. physische Adressen
  - Adressübersetzung
  - Seitentabellen
- Swapping

CPU/Hardware-Ebene

- Segementierung/Paging und virtueller Speicher
  - Memory Management Unit + Translation Lookaside Buffer



/1/

CPU/Hardware-Ebene

- Speicherarten: /5/

Typ	flüchtig?	schreiben?	Löschgröße	Löschzyklen	Kosten/byte	Gewindigkeit
SRAM	ja	ja	Byte	unbegrenzt	teuer	schnell
DRAM	ja	ja	Byte	unbegrenzt	mittel	mittel
ROM	nein	nein	-	-	billig	schnell
PROM	nein	einmal, durch Programmierer	-	-	mittel	mittel
EPROM	nein	ja, durch Programmierer	ganzer Chip	begrenzt	mittel	schnell
EEPROM	nein	ja	Byte	begrenzt	teuer	schnell zu lesen, langsam zu schreiben
Flash	nein	ja	Sektor	begrenzt	mittel	schnell zu lesen, langsam zu schreiben
NVRAM	nein	ja	Byte	unbegrenzt	teuer	schnell

Betriebssystemebene

- Herausforderungen für eingebettete Systeme:
  - Speicherbereitstellung und –zugriff langsam, nicht deterministisch
    - Speicherlatenz
    - Swapping
  - oft keine Hardware-Unterstützung (MMU, TLB)
  - wenig Speicher, kein Sekundärspeicher

- Ziel: schneller und deterministische Speicherverwaltung

Betriebssystemebene

- Speicher sperren
  - Swapping macht Speicherzugriffe nicht vorhersagbar → Swapping deaktivieren
  - WindowsNT
    - Flag in `CreateThread()`
  - POSIX
    - `mlockall(), munlockall()`
      → alle Seiten eines Tasks
    - `mlock(), munlock()`
      → bestimmter bereitgestellter Speicherbereich

Betriebssystemebene

- Speicherabbildung
  - Abbildung logischer Adressen auf physikalische Adressen
  - Abbildung auf z.B.
    - Anbindung an Peripheriegeräte
    - Dateien
    - gemeinsamer Speicher /Interprozesskommunikation
  - POSIX:
    - `mmap()` → gibt logische Adresse für physk. Adr. zurück
    - `mprotect()` → Zugriffssteuerung

Virtuelle Maschinen (Laufzeitumg.)

- Probleme:
  - Garbage Collection
  - dynamische Speicherbereitstellung

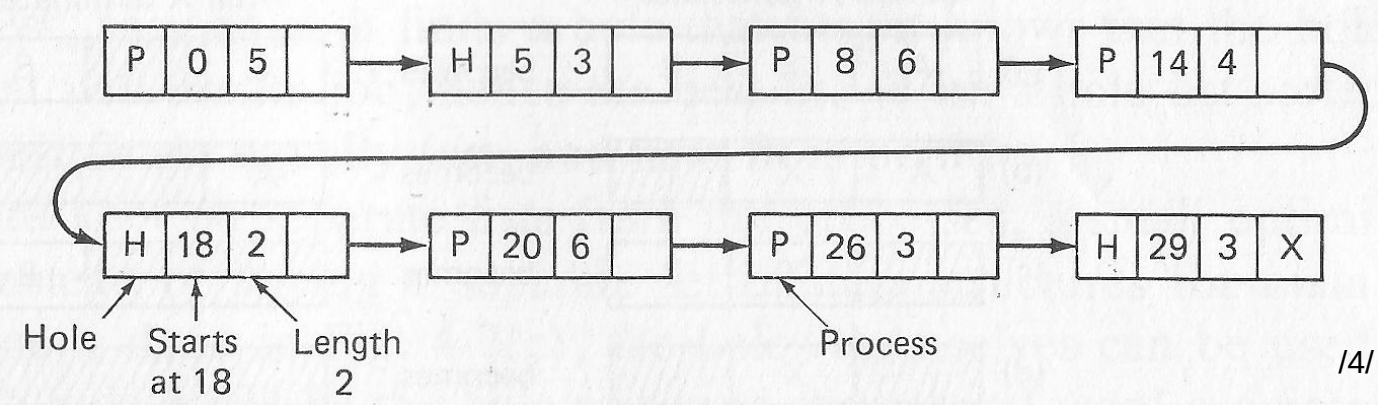
- Bsp. RT Java:
  - verschiedene Speicherarten
    - Heap mit GC
    - „unsterblicher" Speicher
    - Speicher mit begrenzter Lebensdauer (Speicherbereitstellung linear oder variabel)

Algorithmen für Speicherbereitstellung

- statische Speicherbereitstellung
  - Segmentierung, feste Größe
  - schnell und vorhersagbar, aber Größe nicht änderbar
- dynamische Speicherbereitstellung
  - Änderung zur Laufzeit
  - „Stück“ aus globalem Heap
  - Überwachung des freien und belegten Speichers
  - verkettete Listen, Bitmaps, Buddy-Systeme, getrennte Free-Listen

dynamische Speicherbereitstellung – verkettete Liste

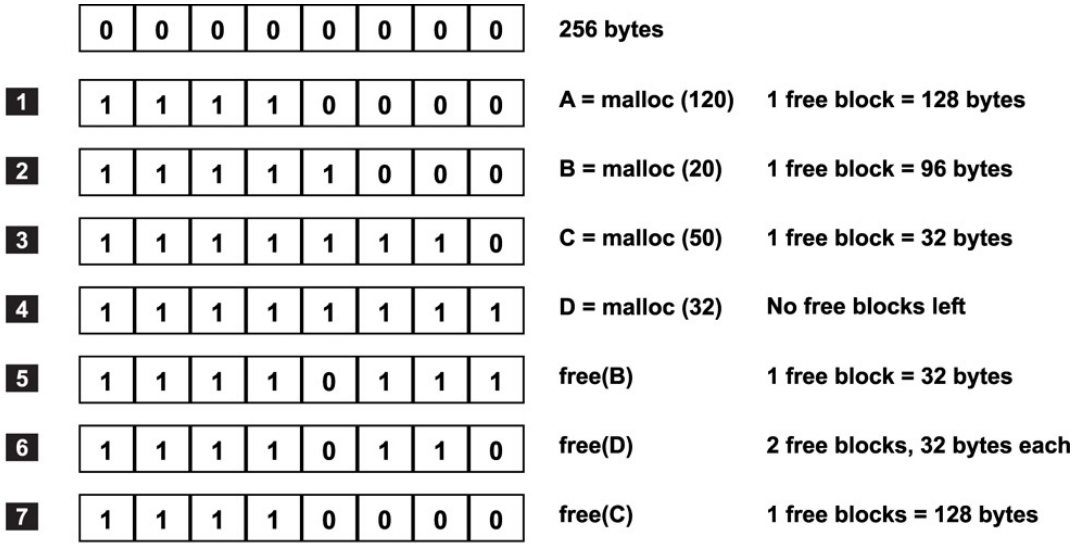
- alle vergebenen freien Blöcke in einer verketten Liste
- Bereitstellen eines neuen Blocks:
  - durch Liste iterieren
  - first-fit/next-fit, best-fit, worst-fit



/4/

dynamische Speicherbereitstellung – Bitmap/Array

- Aufteilung des Speichers in gleich große Teile
- Bitmap gibt an, ob vergeben (1) oder frei (0)
- Iteration über Bitmap

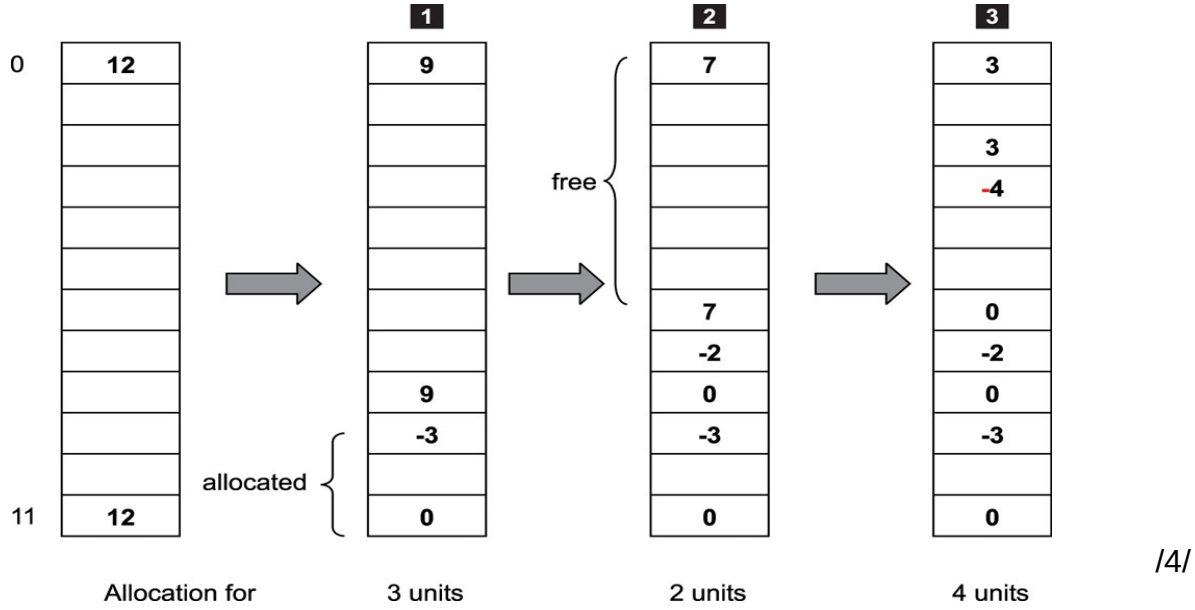


/4/

- Problem: Fragmentierung

dynamische Speicherbereitstellung – Bitmap/Array

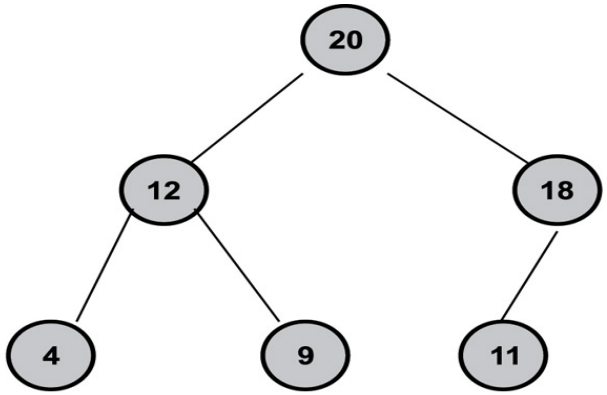
- Modifikation: anderes Kodierungsschema
  - positive Zahl an Anfang und Ende von freien Speicherbereich
  - negative Zahl am Anfang (-1xGröße), 0 am Ende von belegten Speicherbereichen

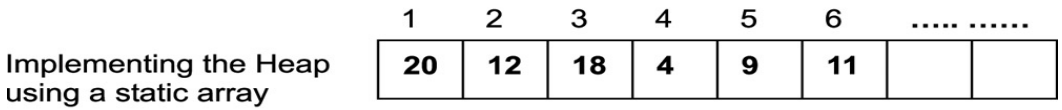


/4/

dynamische Speicherbereitstellung – Suchen von Speicherblöcken

- finden eines passend großen Bereichs freien Speichers durch Iteratrion
  $\rightarrow$ bessere Lösung ist sortieren
- Heap-Datenstruktur:





The left child of a node k is at position 2k.
The right child of a node k is at position 2k+1.

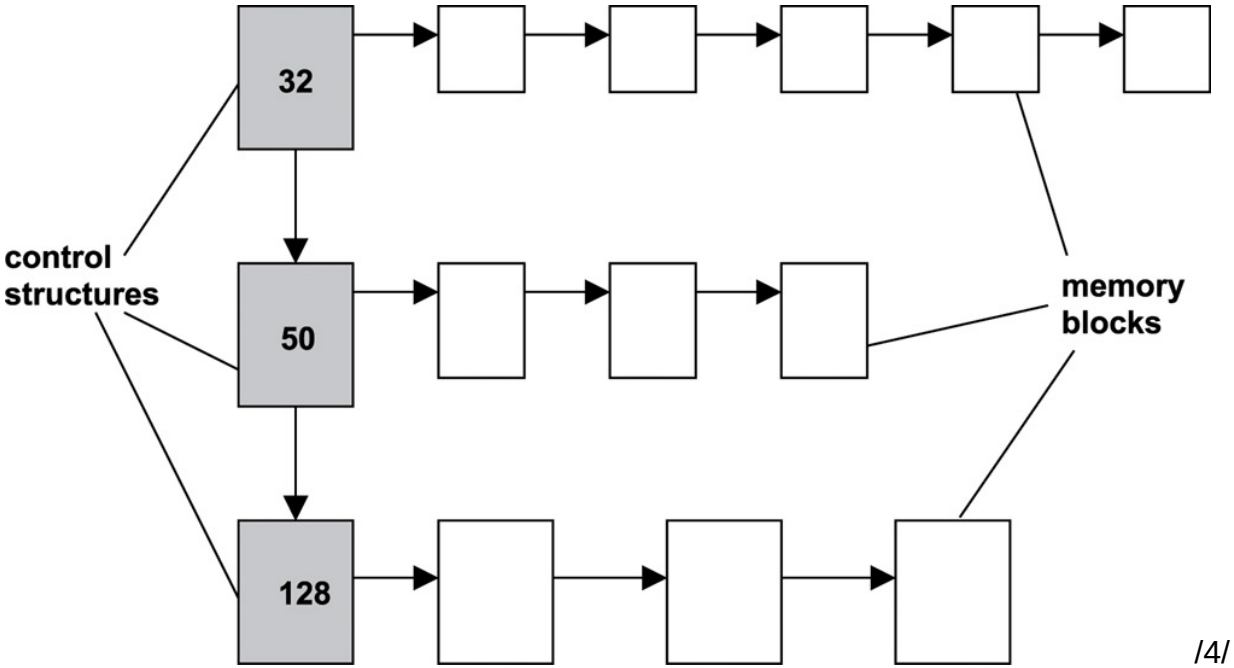
/4/

dynamische Speicherbereitstellung – Speicher freigeben

- zugeteilter Speicher kann auch wieder freigegeben werden
- Herausforderungen:
  - Freispeicherblöcke zusammenfassen
  - Fragmentierung verhindern/beseitigen

dynamische Speicherbereitstellung – Speicherpools

- Unterteilung des Speichers in mehrere Speicherpools mit festen Blockgrößen
- besserer Vorhersagbarkeit

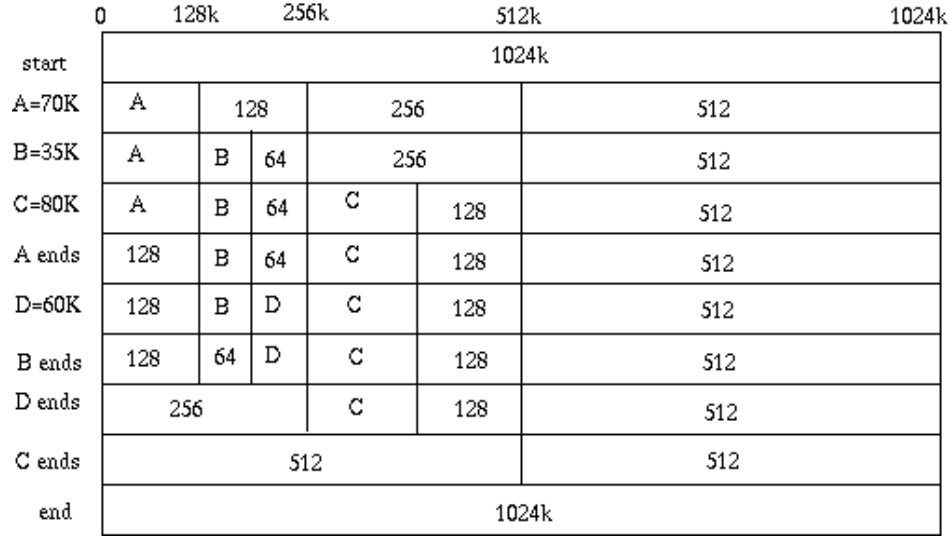


/4/

dynamische Speicherbereitstellung – Buddy-Verfahren

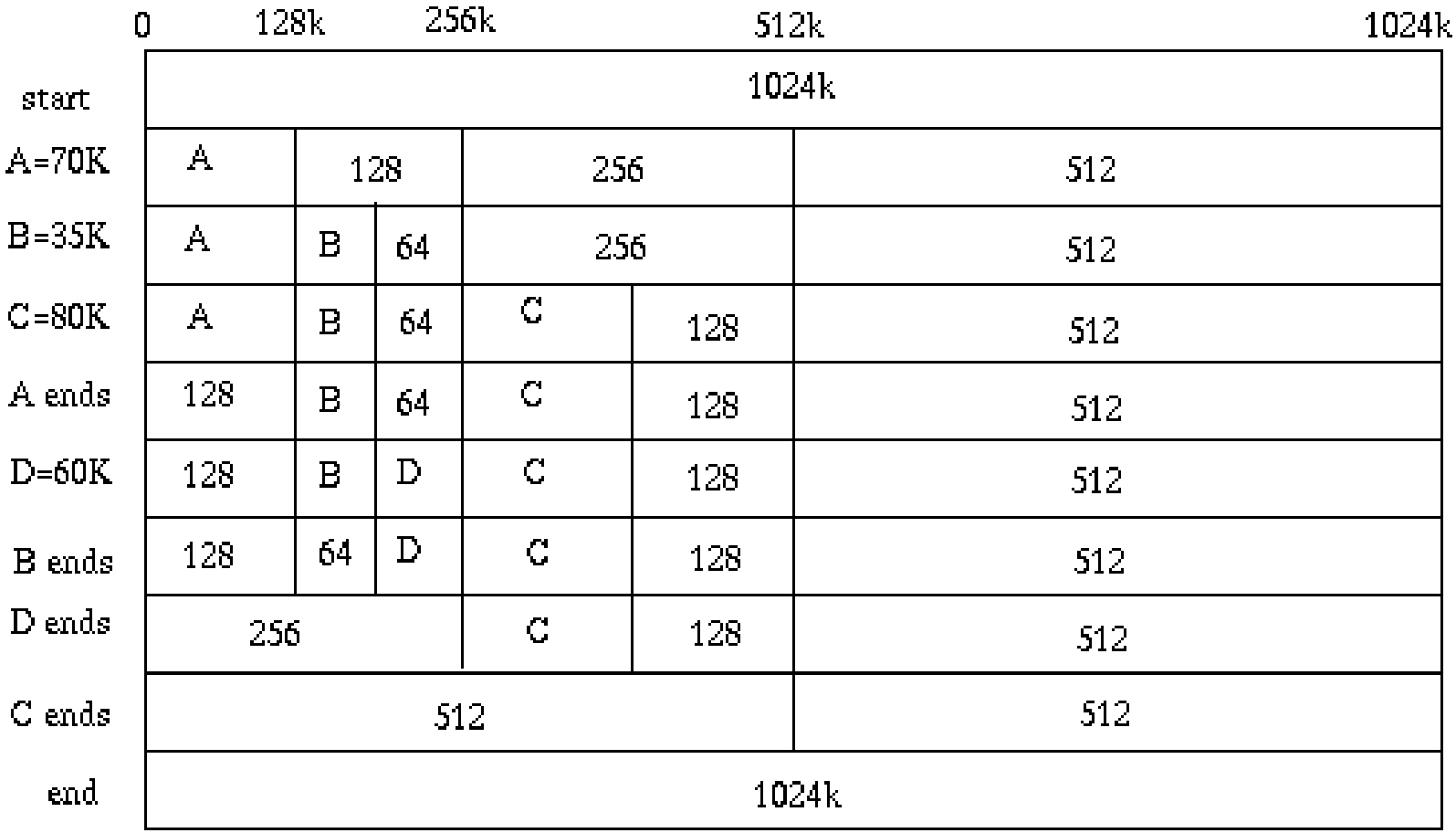
1. Speicher wird in Bereiche mit Länge 2^K aufgeteilt
2. angeforderte Speichergröße wird zur nächsten 2er-Potenz aufgerundet
3. Speicherbereich dieser Größe wird gesucht
4. Nicht gefunden → doppelte Größe gesucht und dieser geteilt
5. (rekursiv weiter)

- Vorteil:
  begrenzte Laufzeit,
  leicht zu implementieren
- Nachteil:
  hohe Fragmentierung



/4/

dynamische Speicherbereitstellung – Buddy-Verfahren



/4/

Fragen?

- Speicherverwaltung
  - Grundlagen
  - Möglichkeiten auf verschiedenen Ebenen
    - CPU/Hardware, Betriebssystem, virtuelle Maschinen
  - Algorithmen zur Speicherbereitstellung
    - statische vs. dynamische Bereitstellung
    - verkettete Listen, Bitmaps/Vektoren, Buddy-Systeme

Hasso Plattner Institut

IT Systems Engineering | Universität Potsdam

Interrupts & Exceptions

Grundlagen

- eingebettete Systeme reagieren auf externe Ereignisse

- zyklische Abfrage (Polling)
  - schnell
  - Overhead, wenn sich Daten nicht ändern
- Interrupts
  - langsamere Reaktionszeit, Abarbeitung dauert länger
  - effizienter → Daten werden verarbeitet, wenn verfügbar
  - aber: nicht deterministisch, wenn Auftrittsrate nicht bekannt

Interruptbehandlung

1. CPU prüft Interrupt-Leitung vor jeder Instruktionsausführung
2. Wenn Interrupt signalisiert: nächste Instruktion nicht ausführen, sondern Befehlszähler (etc.) sichern
3. Befehlszähler wird auf Adresse für Interruptbehandlungsroutine gesetzt (Interruptvektor; fest oder aus Interrupttabelle)
4. Ausführung (Interrupt Service Routine)
5. Zurücksetzten des Programmzählers (etc.)

Probleme

- mehrere Interrupts gleichzeitig, ISR löst selbst Interrupt aus

  → Priorisierung, Maskierung

→ Programmable Interrupt Controller

  - durch Software konfigurierbare Hardwareeinheit
  - Prioritäten, Maskierung, Warteschlangen
  - Advanced PIC: Interruptverteilung in Multiprozessorsystemen

- Interrupt-Latenz
  - Hardwarelatenz
  - Interrupts höherer Priorität
  - Kontextwechsel

Exceptions

- synchrone Interrupts (ausgelöst durch Instruktionsausführung)

Typ	**Grund**	**Folge**	**Beispiel**
Interrupt	*I/O-Signal*	*Rückkehr zur nächsten Anweisung*	*externes Gerät (asynchron)*
Trap	erwartete Ausnahme	Rückkehr zur nächsten Anweisung	Lesen einer Datei
Fault	potentiell behebbarer Fehler	evtl. Rückkehr zur gleichen Anweisung	Seitenfehler
Abort	nicht behebbarer Fehler	kehrt nicht zurück	Hardwarefehler

Fragen?

- Interrupts
  - Interrupt vs. Polling
  - Wie wird ein Interrupt behandelt/durchgeführt
  - Maskierung, Schachtelung, (A)PIC
  - Interrupt-Latenz
- Exceptions

Quellen

1. Operating System Concepts, Seventh Edition; A. Silberschatz, P. B. Galvin, G. Gagne; John Wiley & Sons, Inc.; 2005
2. Inside Microsoft Windows 2000, Third Edition; D. A. Solomon, M. E. Russinovich; Microsoft Press; 2000
3. Memory Management for Embedded Systems; A. Polze, A. Rasche, B. Rabe; Betriebssysteme für Embedded Computing; WS 07/08
4. Real-Time Concepts for Embedded Systems; Q. Li,C. Yao; Elsevier Books; 01.07.2003
5. Introduction to Memory Types; Michael Barr; Embedded Systems Design; 01.05.2000
6. Interrupts and Exeptions; A. Polze, A. Rasche, B. Rabe; Betriebssysteme für Embedded Computing; WS 07/08
7. Storage Allocation for Real-Time, Embedded Systems; S. M. Donahue, M. P. Hampton, M. Deters, J. M. Nye, R. Cytron, K. M. Kavi; Springer-Verlag; 2001
8. Memory Management for Real-Time Java: An Efficient Solution using Hardware Support; T. Higuera, V. Issarny, M. Banâtre, F. Parain; Springer Netherlands; 30.10.2004