1 Mehrbenutzersysteme

Urspruenglich waren Systeme fuer einen Benutzer gedacht:

- nur ein Programm zu einer Zeit
- keine Vernetzung/ entfernter Zugriff

Heutige Systeme:

- mehrere unabhaengige Programme im Speicher
- unter Umstaenden gleichzeitige Ausfuehrung

Voraussetzung:

- Schutz der Daten zwischen Programmen
- Moeglichkeit der Umschaltung zwischen Programmen

→ Aufgabe des OS

Benoetigt Unterstuetzung aus der Hardware:

- Privilegierter Modus (hier arbeitet nur das OS)
- Interrupts/Fehlerindikatoren
- Speicherverwaltung

1.1 Privilegierte Ausfuehrung

Benutzersysteme sollten nicht direkt auf gemeinsame Hardwareressourcen und Systemkonfiguration zugreifen duerfen

→ Koordination durch OS erforderlich
→ Technische Basis: Ausfuehrungsmodi

Systemmodus:

- voller (=privilegierter) Zugriff auf alle Rechnerkomponenten
- fuer das OS

Benutzermodus:

- eingeschraenkter Zugriff
- keine privilegierten Befehle
- kein Zugriff auf Konfigurationsregister

Moderne Prozessoren haben noch mehr "privilegierte Modi". Diese koennen selbst fuer das OS nicht sicherbar sein, da es z.B. "Firmware" Zugriff sein kann.

1.2 Moduswechsel

Wechsel von System- in Benutzermodus:

- privilegierter Befehl (oft: Schreiben auf ein Kontrollregister)
- OS nutzt das zu Programmstart

Wechsel von Benutzer- in Systemmodus:

- muss kontrolliert werden
- keine Ausfuehrung beliebigen Codes moeglich

→ Realisierung durch speziellen Befehl (SysCall/Trap)
→ Wichtig dabei Sicherung:

→ Sicherung der Ruecksprungadresse
→ Springen an vordefinierte Adresse im OS
→ Umschalten auf Systemmodus

...

→ Rueckkehrbefehl schaltet wieder in Benutzermodus

Interrups/Exceptions fuehren zu einem Wechsel in den privilegierten Modus

1.3 Interrupts

Interrupts erlauben dem Prozessor auf externe, asynchrone Ereignisse zu reagieren

- spezielles Eingangssignal: Unterbrechungsaufforderung
- Moeglichkeit der Maskierung mancher Signale → Ignorieren moeglich

Vorgehen:

→ Signalabfragung am Ende des Befehlszyklus (u.U. Signalleitung von aussen)
→ Falls Signal gesetzt:
→ Sichern des BZ auf dem Stack
→ Sprung an vordefinierte Adresse im OS ("Interrupt Handler")
→ Ruecksprung nach Ende an alte Ausfuehrungsstelle

1.4 Exceptions

Ausnahmen (durch Ausfuehrung einer unzulaessigen Operation) - Traps/Exceptions:

- nicht erlaubte Befehle
- Arithmetische Fehler
- Pagefaults/Segfaults (Speicherzugriffsfehler)
- in Hardware naicht implementierte Befehle

Vorgehen:

→ Abbruch des gerade ausgefuehrten Befehls
→ Sichern des Prozessorzustands (vor allem der BZ)
→ Sprung zu einer vordefinierten Adresse im Systemmodus

Die Exception-Vektortabelle speichert die Adressen der Behandlungsroutinen fuer die verschiedenen Klassen von Ausnahmen

Moegliche Behandlung der Ausnahme im OS:

- Abbruch des Prozesses
- Emulieren der Operation
- Weitergabe an User zur Korrektur

→ falls sinnvoll: Wiederholung der Fortfuehrung des unterbrochenen Befehls

1.5 Interrupts vs. Exceptions

Interrupts sind asynchron
→ sie werden ausgeloest durch externe Ereignisse (koennen nach beliebigen Befehlen auftreten) und sind in der Regel Teil des normalen Betriebs

Exceptions sind synchron
→ sie signalisieren in der Regel einen Fehler in der Ausfuehrung, treten dabei nach der Abarbeitung eines Befehls auf und wirken nach diesem direkt

1.6 Speicherverwaltung

Mehrere Programme im Speicher
→ implementiert als Prozess (Programm + Daten + Kontext + Stack)

Das Programm schaltet zwischen Prozessen um:

→ Unterbrechung der Programmausfuehrung (durch Interrupts)
→ Speichern des alten BZ und aller Register
→ Scheduling: Auswahl des naechsten Programmes
→ Laden des neuen BZ und aller zugehoerigen Register

Konsequenzen:

- kein Schutz zwischen Prozessen
- jeder Prozess sieht alle Daten
- Prozess koennte OS aendern
- ineffiziente Speichernutzung

1.6.1 Naiver Ansatz

Schutzmechanismus:

- Zugriffsbeschraenkung noetig (nur auf eigenen Bereich)
- geregelt durch z.B. Spezialregister (min/max Adresse) → sonst Exception

Privilegierter Modus:

- alle Benutzerprogramme laufen im "User-Mode"
- Setzen von min/max Adresse ist privilegiert

Nachteile:

- sehr grobgranular → ineffiziente Speichernutzung
- Prozesse an variierenden Startadressen

1.6.2 mehrere Adressraeume

Beliebig viele "virtuelle" Adressraeume
→ jeder Prozess sieht/bekommt nur seinen eigenen Raum (Isolation)

Betriebssystem schaltet Adressraeume um
→ waehrend des Scheduling - Adressraum wird Teil des Protesses

- weiterhin physikalischer Speicher
- effektive Adressen sind virutelle Adressen
- Zugriff im Speicher ueber physikalische Adressen (Umrechnung durch CPU)

1.7 Uebersetzung virtueller Adressen

Adressen mussen bei jedem Zugriff uebersetzt werden
→ "Memory Management Unit" (MMU)

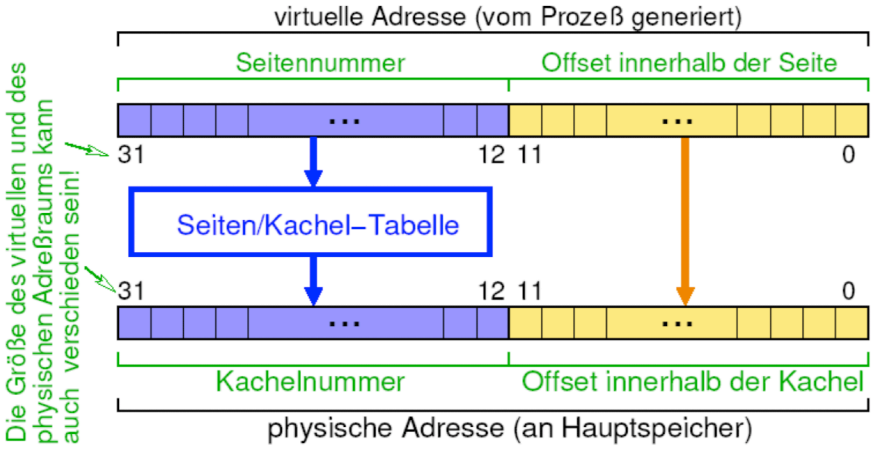
MMU:

- Uebersetzt jede angeforderte Adresse
- Zugriff auf physikalische Adresse im Speicher → Programme haben keinen direkten Zugriff mehr
- Uebersetzung wird durch OS kontrolliert

1.8 MMU Design

OS verwaltet physikalischen Speicher (nicht mehr kontinuierlicher Speicher)

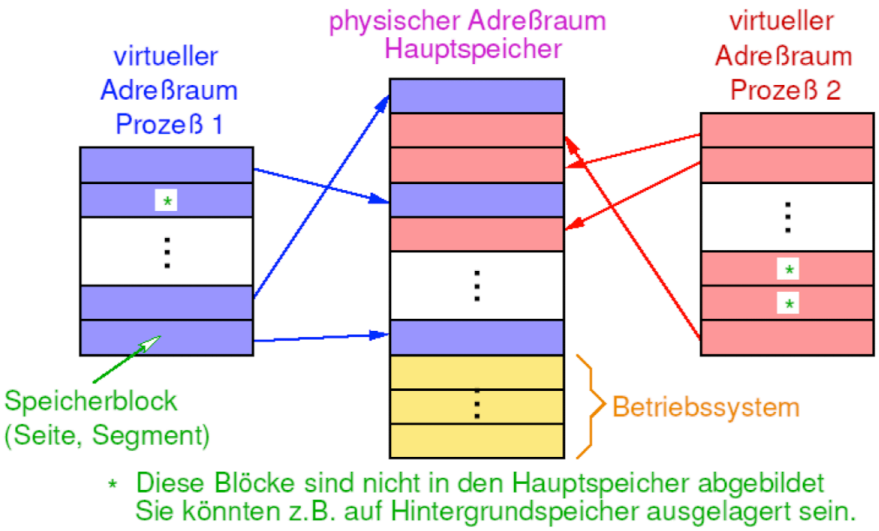
- virtuelle Adressen werden auf physikalischen Speicher abgebildet
- einzelne Prozesse sehen eine Untermenge vom Speicher
- Reihung von kontinuierlichen Adressen (Pages), da Tabelle sonst zu gross
- Pagegroesse oft 4 KiB



Zuordnung:

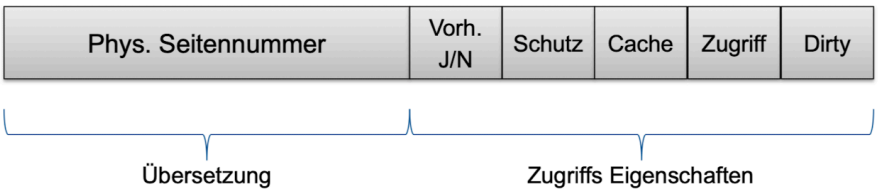
- von Seiten zu Kacheln
- von virtuellen Seiten zu physikalischen Seiten ("Frames")
- direkte Uebersetzung innehalb der Page

1.9 Speicherabbildung



Implementiert durch Seitentabellen:

- Eintrag fuer jede viruelle Seite
- Enthaelt Seitennummer der physikalischen Seite



- physikalische Seitenummer: wird addiert zu Adressoffset
- vorhanden Ja/Nein: Bit signalisiert ob Eintrag gueltig ist
- Schutzbits: Read/Write erlaubt, bzw. privat J/N
- Zugriff: Page accessed J/N
- Dirty: Page written J/N

1.10 Uebersetzung

Zugriff auf Adresse A als Lese und Schreibevorgang:

→ Zerlegung in A_{Seite} (obere Bits) und A_{Offset} (untere Bits)
→ Lesen der Basisadresse der Page (Ueblicherweise privilegiertes Register)
→ Eintragsberechnung A_{Seite}
→ Auslesen des Page Eintrags:

- → nicht gueltiger Eintrag → Exception
- → privilegierte Seite in Usermode accessed → Exception
- → Write access aber nur Read erlaubt → Exception

→ Physikalische Adresse = Page Eintrag + A_{Offset}
→ Zugriff auf physikalische Adresse

2 IA-32 Speicherverwaltung

Programme nutzen virtuelle Adressen:

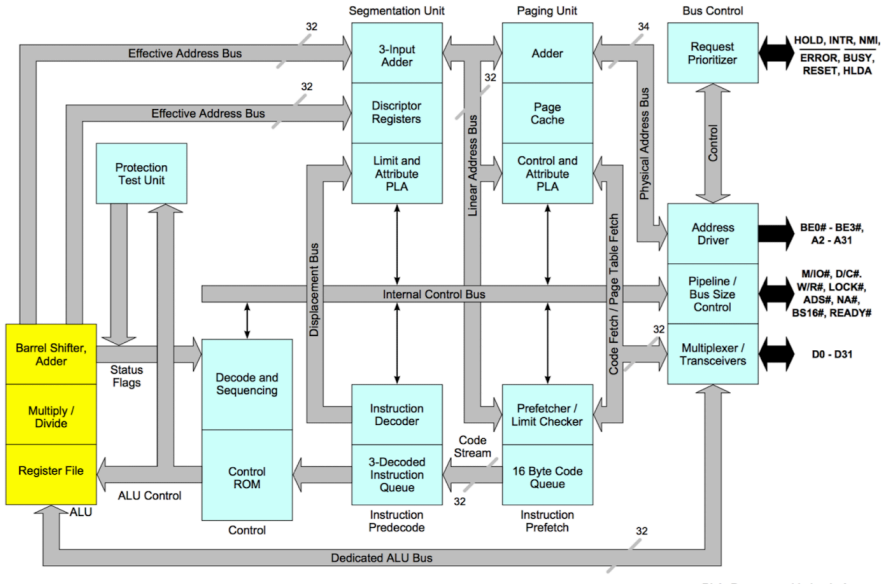
$$Basisreg + (Indexreg \cdot Skalar) + Displacement$$

Umwandlung in physikalische Adresse:

- Seitenbasierte Umsetzung (Seitengroesse 4 KiB (Bit 0-11 als Offset))
- zweistufige Seitentabelle:

  - Bit 31-22: Directory Eintrag
  - Bit 21-12: Page Table Eintrag

- privilegiertes "PD" Register zeigt auf Directory



Herausforderungen:

- Hardwarekomplexitaet (teilweise auf Software ausgelagert/ Exceptions)
- Pages koennen sehr gross werden (viele ungenutzt, aber viele noetig)

→ groessere Seiten ("Large Pages") und mehrstufige Unterteilung dieser
→ Ermoeglichen von schnelleren Zugriff auf kuerzlich verwendete Pages

2.1 Speicherverwaltung von Virtuellen Speichern

Neuer Speicher fuer Daten:

- beim Laden eines Programmes
- dynamisch im Stack/Heap

Speicherverwaltung im Usermodus:

- maximale Verfuegbarkeit vom OS anfordern
- restliche Verwaltung im Laufzeitsystem (Vergabe von virtuellen Adressen)

Abbildung auf physikalischen Speicher:

- bei maximaler Forderung → alle Seiten physikalisch einrichten
- sonst Lazy → Erzeugung von Seiten erst falls sie gebraucht werden
- falls kein physikalischer Speicher frei → Seitenaustausch mit lang nicht verwendeter Page (Legen der alten Page auf Hintergrundspeicher) → Paging

Konsequenzen der Speicherauslegung:

- Teile des Hintergrundspeichers werden als Hauptspeicher genutzt
- Hintergrundspeicher transparent fuer die Anwendung

→ Performanznachteil (Hintergrundspeicher viel langsamer)
→ OS sollte nicht ausgelagert werden (Pinnen von wichtigen Pages)
→ keine Auslagerung durch Pinning

Zugriffsschutz (durch Bits) → geziehlte Fehlerbehandung (Exception Throwing)