Principper for Samtidighed og Styresystemer

Memory Management

René Rydhof Hansen

Marts 2008

Husk!

- Forelæsning tirsdag den 1. april 2008 er rykket til 10:00–10:45.
- The Great OS Shoot Out: tirsdag den 1. april 2008 11:00–12:00 og 14:30–16:16.

Opgaver

- Opgave 1: Project 4.7 (Deadlock monitor)
- Opgave 2: Banker's Algorithm

Memory Management: Oversigt

- Simpel lagerallokering i primær lager
- Paging og segmentering
- Organisation af sidetabel
- Demand paging og sideerstatningsalgoritmer
- Rammeallokering

Lagerhierarkiet

- Registers (< 1 cycle)
- Internal cache (1 cycle)
- Secondary (external) cache (5 cycles)
- Tertiery cache (if any) (10 cycles)
- Physical memory (25–50 cycles)
- Swap disks and file system disks (virtual memory) ($\sim$ 1.000.000 cycles)

Udfordringer

- Hvordan placeres processer i primær lager?
- Hvordan isoleres processer fra hinanden?
- Hvordan isoleres styresystemet fra processer?
- Hvordan deles primær lager mellem processer?
  - Interprocesskommunikation (IPC)
  - Deling af programtekst
- Hvordan implementeres swapping?
- Hvordan undgås fragmentering af primær lager?

Udfordringer

- Hvordan placeres processer i primær lager?
- Hvordan isoleres processer fra hinanden?
- Hvordan isoleres styresystemet fra processer?
- Hvordan deles primær lager mellem processer?
  - Interprocesskommunikation (IPC)
  - Deling af programtekst
- Hvordan implementeres swapping?
- Hvordan undgås fragmentering af primær lager?

Virtuelt lager!

Udfordringer: Uden virtuelt lager

Uden virtuelt lager:

- Alle processer ligger i et fælles adresserum
- Alle lagerreferencer i programteksten er absolutte
- Styresystemet må ved oprettelse af nye processer omskrive alle lagerreferencer til unikke adresser

- Programtekst kan ikke deles mellem processer
- Hvordan beskyttes en proces's data mod at blive læst af en anden process?

Virtuelt lager

- Et adresserum er en mængde adresser
- CPU-addreserbare adresser danner et adresserum
- Virtuelt lager: hver process tildeles et eget adresserum
- Virtuelt lager kan være større end primær lager
- Adresse 0 (eller X) i to forskellige adresserum refererer til forskellige fysiske adresser
- Virtuelt adresserum typisk på størrelse med CPU'ens adresserum

Example

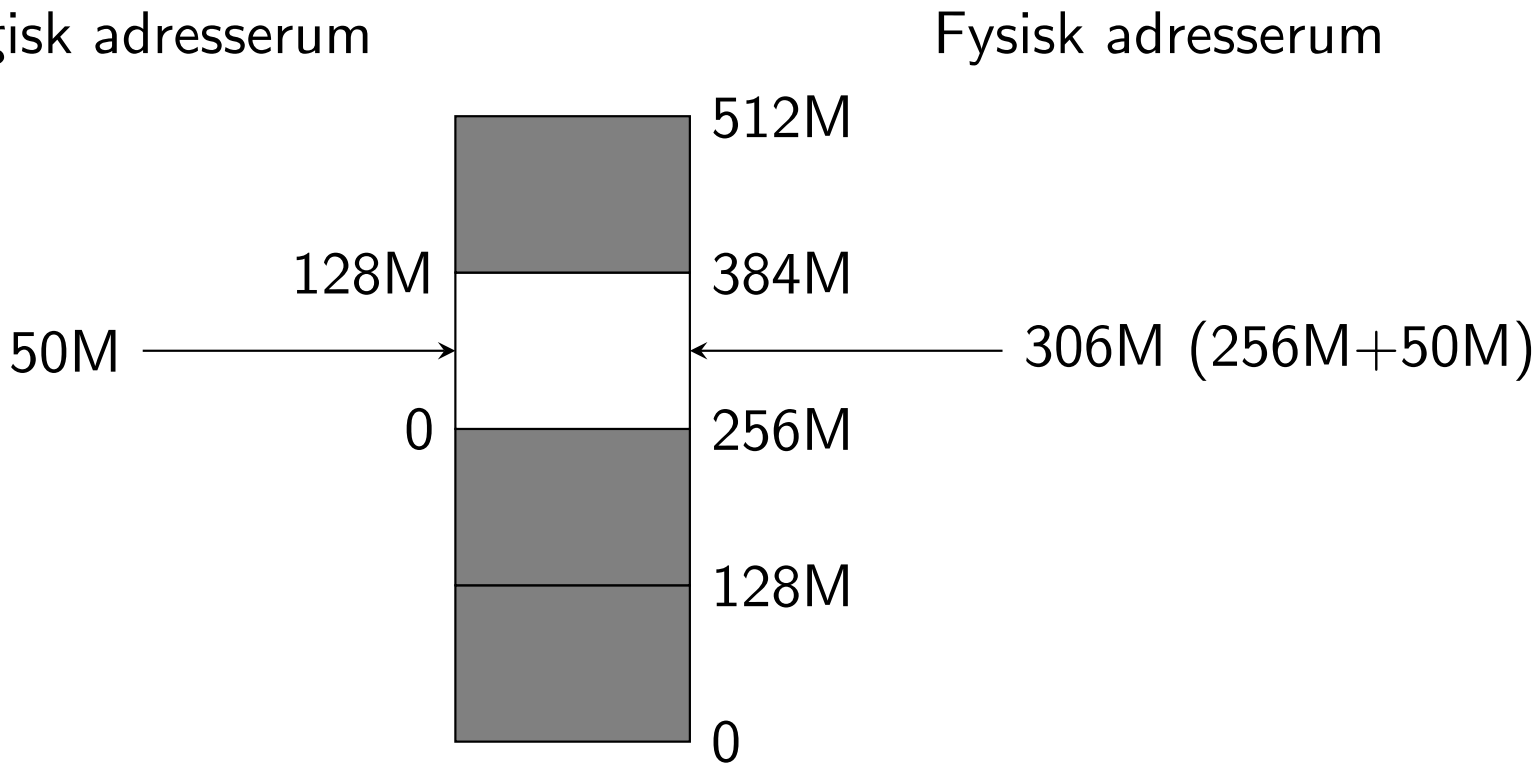
Pentium bruger 32 bits adresseing, dvs. adresser fra 0 til 4GB-1.

Adressetyper

- Fysiske adresser (aka: absolutte adresser) angiver en specifik fysisk lagercelle i det primære lager
- Virtuelle adresser (aka: logiske adresser) er referencer til en lagerlokation uafhængig af den fysiske placering
  - Indirection
  - Logiske adresser skal på køretid afbildes til fysiske adresser
  - "Every software problem can be solved by adding another layer of indirection" (Steven M. Bellovin)
- Relative adresser er defineret i forhold til en kendt basisadresse
  - Et eksempel på en logisk adresse
  - Basisadressen er en kendt fysisk adresse
  - Den relative adresse er specificeret som et offset i forhold til basisadressen
  - Den fysiske adresse beregnes som summen af basisadresse og offset

Adresseoversættelse

- Logiske adresser oversættes på kørselstidspunktet til fysiske adresser
  - Kræver en "memory management unit" (MMU)
  - Hardware ofte integreret i CPU'en
  - Virtuelt lager er transparent for brugeren



Adresseoversættelse

```
a = a + b

load r1,0       ; læs celle 0 til register r1
load r2,4       ; læs celle 4 til register r2
add r1,r2       ; r1 = r1 + r2
store 0,r1      ; gem resultat i celle 0
```

- Hvis 0 og 4 er logiske adresser oversættes de af MMU'en til fysiske adresser
- For forskellige processer vil dette være forskellige fysiske adresser
- Muliggør deling af programtekst mellem processer

Memory Management: Oversigt

- Simpel lagerallokering i primær lager
- Paging og segmentering
- Organisation af sidetabel
- Demand paging og sideerstatningsalgoritmer
- Rammeallokering

Fast inddeling

- Inddeling af primær lager i et antal blokke af fast størrelse
- Ens størrelse
  - Eksempel: 4x128MB = 512MB
- Uens størrelse
  - Eksempel: 128MB+2x64MB+4x32MB+8x16MB = 512MB
  - Kræver strategi for tildeling af processer til blokke
  - Mindste passende
    - Kø af processer indtil passende blok er ledig
  - Mindste ledige
    - Mere intern fragmentering
- Intern fragmentering er ubrugt lager i en blok

Dynamisk inddeling

- Processer angiver lagerbehov ved opstart
  - Styresystemet allokerer passende blok
  - Styresystemet vedligeholder startadressen og bloklængden
- Fordele
  - Ingen (synlig) intern fragmentering
  - Tilpasser sig aktuelt behov (få store eller mange små processer)
- Ulemper
  - Kræver at lagerbehov er kendt i forvejen
  - Ekstern fragmentering er ubrugt lager mellem blokkene
  - Kræver strategi for tildeling af processer til blokke
  - Compaction: teknik til relokering af processer for at reducere ekstern fragmentering
  - Compaction uden virtuelt lager kræver omskrivning af adresser i programtekst og data (pointere)

Memory Management: Oversigt

- Simpel lagerallokering i primær lager
- Paging og segmentering
- Organisation af sidetabel
- Demand paging og sideerstatningsalgoritmer
- Rammeallokering

Paging

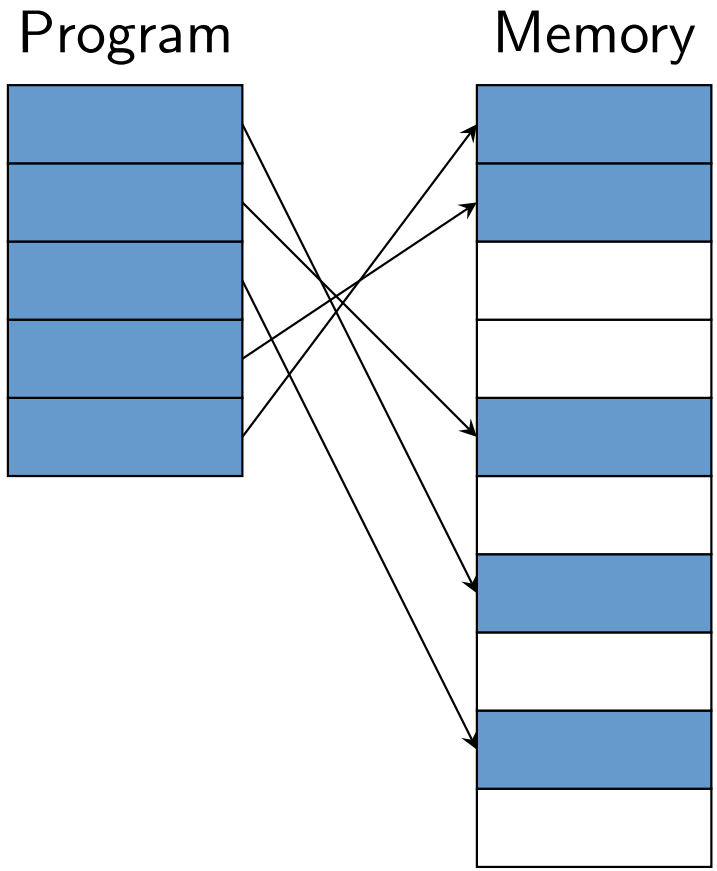
- Det virtuelle lager opdeles i et antal sider (pages) af fast størrelse
- Det fysiske lager opdeles i et antal rammer (page frames)
  - Sider placeres i rammer
  - Rammer har samme faste størrelse som sider
- Styresystemet indplacerer sider i rammer og vedligeholder informationer om dette i en sidetabel

Example

På en Pentium er sider typisk 4KB og på en Sparc typisk 8KB store.

Fragmentering ved paging

- Ingen ekstern fragmentering
- Minimal intern fragmentering



Adresseoversættelse ved paging

- Logiske adresser er på formen $\langle \text{sidenr (20bit)}, \text{offset (12bit)} \rangle$
- Fysiske adresser er på formen $\langle \text{rammenr}, \text{offset} \rangle$
- Antag 4KB sider på en 32-bit maskine:
  - Side 0 er fra 0x00000000 til 0x00000FFF
  - Side 1 er fra 0x00001000 til 0x00001FFF
  - Side 2 er fra 0x00002000 til 0x00002FFF
- Sidetabellen afbilder sidenumre til rammenummer:

$$\text{fysisk adr.} = \text{sidetabel}[\text{sidenr}].\text{rammenr} \cdot \text{sidestørrelse} + \text{offset}$$

Segmentering

- Hukommelsen inddeles i segmenter:
  - Et eget adresserum
  - Egen størrelse, adgangsrettigheder, etc
  - Defineret ved startadresse (basisadresse) og længde
  - Logisk adresse er på formen $\langle \text{sgementnr}, \text{offset} \rangle$
  - Segmenttabellen afbilder segmentnumre til startadressen og segmentlængden
  - Fysisk adresse (hvis $0 \leq \text{offset} < \text{segmenttabel[segmentnr].length}$:

$$\text{fysisk adr.} = \text{segmenttabel[segmentnr].start} + \text{offset}$$

Segmentering: Fordele og Ulemper

- Fordele
  - Segmenter defineres af programmøren
    - Gør det muligt at samle data i logiske enheder: programtekst, data, stak, moduler, etc.
  - Finere granularitet af rettigheder
    - Indeksoverløb for arrays allokeret som et segment kan kontrolleres af hardwaren
    - Stakoverløb
  - Ingen intern fragmentering
- Ulemper
  - Placering af segmenter i primær lager
  - Ekstern fragmentering
  - Manglende understøttelse i main-stream OS

Segmentering: Pentium

- Seks 16-bit segmentregistre
  - CS,DS,ES,FS,GS,SS
  - Indeks til segmenttabeller
- Lagertilgang skeer relativt til et segmentregister
  - CS: code segment
  - SS: stack segment
  - Segment vælges som del af instruktion
- Linux og Windows bruger ikke segmentering
  - Alle segmentregistre peger på samme (store) segment

Paging + Segmentering

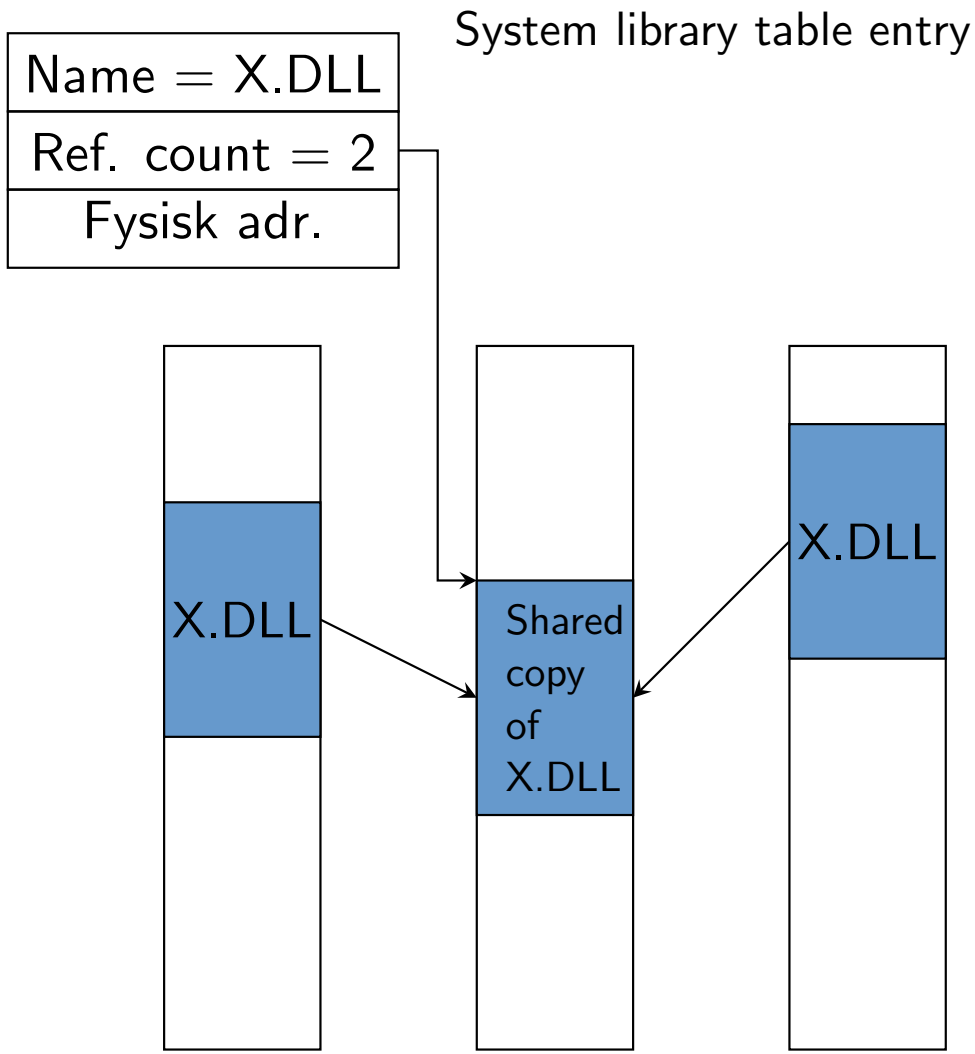
- Logiske adresser er på formen $\langle \text{segmentnr}, \text{sidenr}, \text{offset} \rangle$
- Kræver segmenttabel samt en sidetabel per segment
- Segmenttabellen angiver starten af sidetabellen
- Fysisk adresse:

$$\begin{aligned} \text{fysisk adr.} \quad = \quad & \text{segmenttabel}[\text{segmentnr}].\text{start}[\text{sidenr}].\text{rammenr} \\ & \cdot \text{sidestørrelse} + \text{offset} \end{aligned}$$

Delt hukommelse

- Hukommelse kan deles mellem processer ved at forskellige sidetabeller henviser til samme ramme
- Bruges til
  - Interproceskommunikation
  - Deling af programtekst og biblioteker (DLL, `.so`)
- Også muligt med segmentering
  - To processers segmenttabeller henviser til samme segment
- Samme fysiske lager i to forskellige (processers) adresserum
- Delte rammer behøver ikke give anledning til samme (lokale) sidenummer

Delte biblioteker



Styresystemets hukommelse

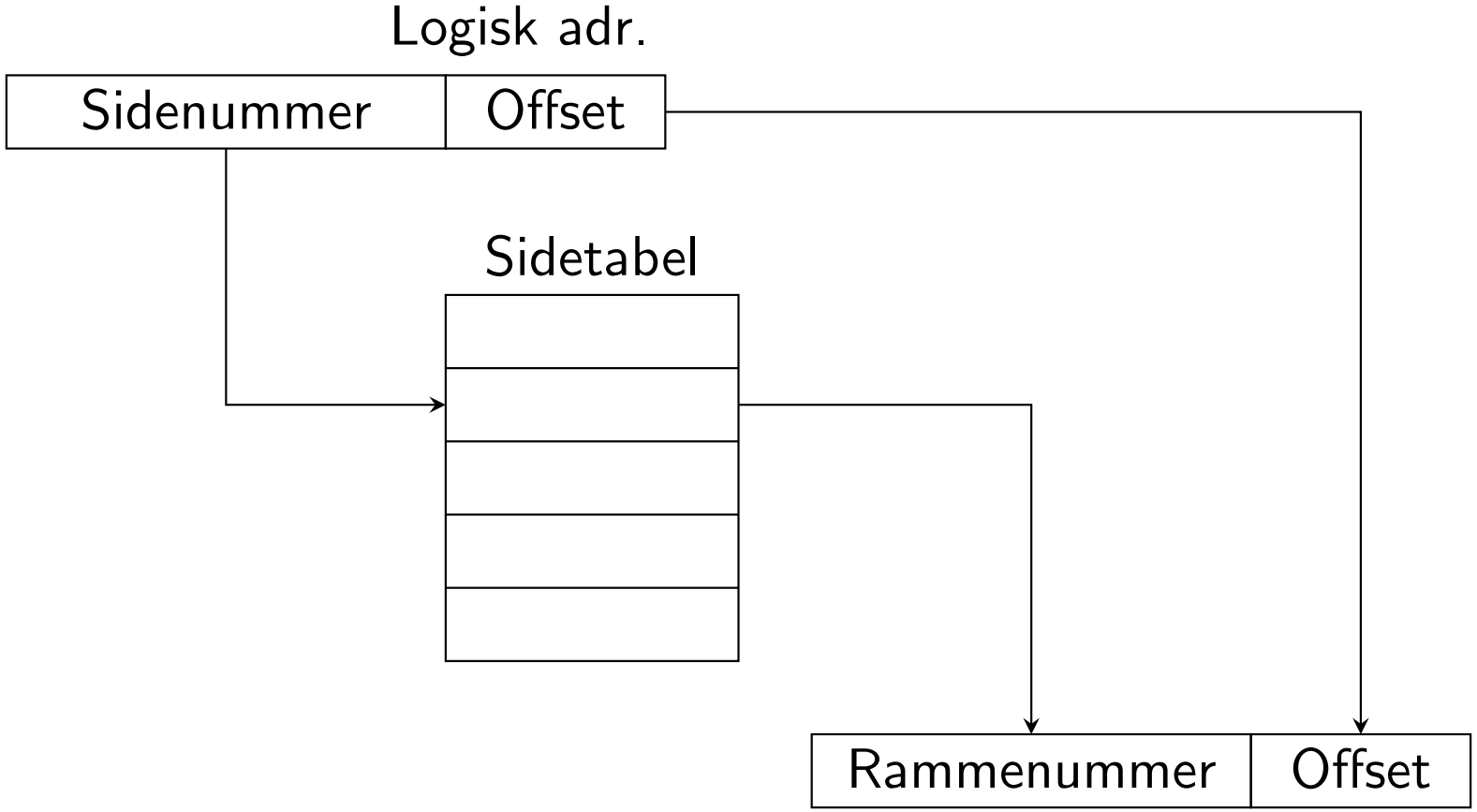
- Styresystemet deles af alle processer
- Styresystemets hukommelse afbildes typisk i procesernes adresserum
  - Windows bruger en 2GB/2GB opdeling
  - Linux bruger en 3GB/1GB opdeling
  - Styresystemet afbildes i øverste del af adresserummet

Memory Management: Oversigt

- Simpel lagerallokering i primær lager
- Paging og segmentering
- Organisation af sidetabel
- Demand paging og sideerstatningsalgoritmer
- Rammeallokering

Direkte sidetabel (et-niveau sidetabel)

- Sidetabellen bruges af MMU'en til adresseoversættelse ved paging
- Vedligeholdes af styresystemet
- Een sidetabel per process med en indgang per side
- *n*'te indgang i sidetabellen giver placering af den *n*'te side i fysisk hukommelse



To-niveau sidetabeller

- Problem: 20-bit sidenumre giver 2^{20} indgange i sidetabellen
  - Med 32 bit per indgang giver dette 4MB — per process!
- Løsning: Del sidetabellen op i flere niveauer
  - Sidenummeret deles i fx 2 gange 10 bit
  - Første 10 bit er indgang til første niveau
  - Næste 10 bit er indgang til andet niveau
- 2^{10} indgange per tabel $=$ 4KB $=$ 1 side per tabel
  - Mulighed for at ikke hele sidetabellen skal allokeres
  - Mulighed for at swappe sidetabellen

Inverterede sidetabeller

- På 64-bit maskiner er to-niveau sidetabeller ikke tilstrækkelige
  - Alpha bruger 43-bit adresser, 3 niveauer med 10 bit til hvert niveau og de resterende 13 bit som offset (8KB sidestørrelse)
  - Ved 64 bit adresser kræves endnu flere niveauer
- Løsning: Inverterede sidetabeller
  - Kun een tabel for hele systemet
  - En indgang per ramme med informationer om hvilken side der gemmes i rammen
  - Besværlig adresseoversættelse: hash tabel over sidetabellen skal gennemsøges først
  - Cache af adresseoversættelser kan øge effektiviteten

Sidetabellen

- Faktisk format for sidetabellen er fastlagt af hardwarearkitekturen
- Indgange i sidetabellen inkluderer
  - Om siden er placeret i en ramme
  - Adresse (i hukommelse eller på disk)
  - Rettigheder (read, write, execute; user eller system)
  - Brugsinformationer (tilgået eller modificeret)
  - Ekstra bits til styresystembrug

Pentium sidetabelindgang

- Page frame number (20 bits)
- Spare (3 bits)
- Global page (1 bit)
- Page size (1 bit)
- Dirty (1 bit)
- Accessed (1 bit)
- Cache disabled (1 bit)
- Write-through (1 bit)
- User/supervisor (1 bit)
- Read/write (1 bit)
- Present (1 bit)

Memory Management: Oversigt

- Simpel lagerallokering i primær lager
- Paging og segmentering
- Organisation af sidetabel
- Demand paging og sideerstatningsalgoritmer
- Rammeallokering

Swapping

- Flyt sider, der ikke bruges så ofte, til sekundært lager (disk)
- Lokalitetsprincippet
- Sideerstatningsalgoritmer
  - Hvilke sider flyttes til sekundært lager?
  - Hvornår flyttes sider til sekundært lager?
    - Når der er behov for plads
    - Tidligere (page buffering)
  - Hvornår hentes sider fra sekundært lager?
    - Når der er behov for dem (demand paging)
    - Tidligere (prepaging)
- Rammeallokering
  - Hvor mange rammer skal der allokeres per process?

Demand paging

- Demand paging: sider indlæses ved første tilgang
- Sider kan i sidetabellen markeres som:
  - Ubrugte: fejl ved tilgang
  - Reserverede: allokeres ved næste tilgang (sidetabellen indeholder informationer om hvor indholdet skal hentes fra fx en fil)
  - Committed: siden er placeret i en ramme
- Sider der ikke bruges bliver aldrig allokeret
- Processer kan være større end fysisk lager
- Intet behov for at kende faktisk lagerbehov

Sidefejl

- Hvis processen tilgår en ubrugt eller reserveret side udlæser MMU'en en sidefejl (page fault)
  - Kontrollen overdrages til en page fault handler i styresystemet
- Ubrugte sider
  - Terminerer processen
- Reserverede sider
  - Kan være blanke datasider
  - Kan være programtekst (eller andet filindhold)
  - Kan være sider, der er blevet swappet ud

Prepaging

- Demand paging indlæser sider efter behov
- Prepaging: sider indlæses før de tilgås
- Mens siden indlæses kan styresystemet skifte til en anden tråd og/eller process
- Gør det nemt at indlæse et program:
  - Sidetabllen initialiseres med ubrugte og reserverede sider
  - Når første instruktion udføres udlæses en page fault og styresystemet indlæser siden (med programtekst) fra disk
  - Andre sider indlæses efter behov

Copy-on-write

- Nemt at implementere
  - Marker side som read-only og copy-on-write
  - MMU'en udløser sidefejl ved forsøg på at skrive til siden (read-only)
  - Page fault handler'en laver en kopi af siden (da den er markeret som copy-on-write)
  - Skrivetilgang gennemføres derefter på kopien
  - Hele sidetabellen skal kopieres (medmindre den kan deles)
- Gør det nemt at implementere `fork`
  - Marker alle processens sider som copy-on-write og lad de to processer dele siderne
  - Siderne kopieres efter behov når/hvis de modificeres

Sidestørrelse og demand paging

- Små sider
  - Bedre lokalitet og dermed få sidefejl
  - Store sidetabeller
- Store sider
  - Mindre god lokalitet og dermed flere sidefejl
  - Mindre sidetabeller
- Meget store sider
  - Få sidefejl, men langvarig I/O
  - Meget intern fragmentering

Sideerstatningsalgoritmer

- Hvis alle rammer er i brug når sidefejl opstår, skal en eksisterende side erstattes
- Hvilke(n) side(r) skal flyttes?
  - Global politik: siderne vælges på tværs af processer
  - Lokal politik: et antal rammer allokeres per process (working set) og sider erstattes på process basis
  - Mål: minimér sidefejl

Page buffering

- Styresystemet holder permanent et minimum af rammer fri
- Periodisk (Linux: swap daemon) udvælges et antal rammer, som kan skrives til disk i klynger
- Sider slettes dog ikke fra primært lager men placeres i en sidebuffer. Hvis siden tilgås igen, fjernes den fra sidebufferen
- Ved sidefejl kan sider fra sidebufferen erstattes
- Kan forbedre sideerstatningsalgoritmen, idet dårlige valg kan "fortrydes"

Optimal sideerstatning (OPT)

- Ideelt erstattes den side der skal forblive ubrugt længst
  - Dette princip kaldes for den "optimale sideerstatningsalgoritme" da den giver anledning til færrest sidefejl
  - Ikke en rigtig algoritme, mere et princip og sammenligningsgrundlag

requested	2	3	2	1	5	2	4	5	3	2	5	2
entry 1	2	2	2	2	2	2	4	4	4	2	2	2
entry 2		3	3	3	3	3	3	3	3	3	3	3
entry 3				1	5	5	5	5	5	5	5	5

Least Recently Used (LRU)

- Lokalitetsprincippet: Nyligt tilgåede sider skal med stor sandsynlighed bruges i nærmeste fremtid
- Derfor: erstat den side der ikke er blevet brugt i længst tid
- Ved processer med god lokalitet giver LRU næsten samme resultat som OPT
- Besværlig at implementere
  - Tidsstempling
    - Ekstra plads i sidetabellen
    - Opdatering ved hver tilgang til primær lager(!)
    - Styresystemet skal finde side med mindste tidsstempel
  - Dobbelt kædet liste
    - Flyt siden til front ved tilgang
    - Kræver plads (to pointere per side)
    - Dyr at opdatere (seks pointere skal opdateres)
- LRU bruges ikke i praksis

LRU: Eksempel

requested	2	3	2	1	5	2	4	5	3	2	5	2
entry 1	2	2	2	2	2	2	2	2	3	3	3	3
entry 2		3	3	3	5	5	5	5	5	5	5	5
entry 3				1	1	1	4	4	4	2	2	2

First-in, First-out (FIFO)

- Erstat siden der har været i lageret længst
- Implementeres med en kædet liste
  - Nye sider indsættes forrest i listen
  - Sidste side i listen erstattes
  - Modsat LRU opdateres listen kun ved sideerstatning og page faults
- Tager ikke højde for lokalitet
- Kan forbedres med page buffering

requested	2	3	2	1	5	2	4	5	3	2	5	2
entry 1	2	2	2	2	5	5	5	5	3	3	3	3
entry 2		3	3	3	3	2	2	2	2	2	5	5
entry 3				1	1	1	4	4	4	4	4	2

Clock-algoritmen

- Også kendt som "not recently used"
- Ekstra bit i sidetabellen sættes af MMU'en ved tilgang til siden
- Ekstra bit fanger lokalitet
- Sider erstattes kun hvis de ikke har været brugt siden "viseren" sidst passerede over siden
- I praksis næsten lige så god som LRU (som er næsten lige så god som OPT)

Modificeret Clock-algoritme

- Foretræk umodificerede sider
  - Kræver endnu en bit (dirty) i sidetabellen som sættes når der skrives til siden
  - Led efter ubrugte sider (siden sidst) og umodificerede sider
  - Led efter ubrugte men modificerede sider, reset "used" bit

Memory Management: Oversigt

- Simpel lagerallokering i primær lager
- Paging og segmentering
- Organisation af sidetabel
- Demand paging og sideerstatningsalgoritmer
- Rammeallokering

Rammeallokering

- Hvordan allokeres fysisk plads (rammer) til processer?
- Hvordan fordeles m rammer til n processer?
- Hvad har en proces behov for?
- Sidefejlsraten for en process er tilnærmelsesvist omvendt proportionalt med antallet af allokerede rammer

Fast rammeallokering

- Giv hver process $\frac{m}{n}$ rammer
  - Lige allokering
  - Men forskellige processer har forskellige behov
- Giv hver process et antal rammer proportionalt med dens sideantal

$$\frac{p_i}{p_1 + p_2 + \cdots + p_n} \times m$$

  - Men nogle processer bruger måske alle dens sider, men sandre kun bruger en lille del
  - Hvordan findes de sider der faktisk bruges?

"Working set"-modellen

Definition (Working set)

En proces' working set, $W(t, \Delta)$, til tiden t med vinduet Δ er mængden af sider en proces har tilgået indenfor de seneste Δ tidsenheder

- Δ fastsættes eksperimentelt
  - For lille: lokalitet fanges ikke
  - For stor: for "meget" lokalitet (overestimering)
- Working set'et ændrer sig over tid
- Problem: Kræver igen at MMU'en skal tidsstemple siderne og at styresystemet skal ordne sider efter tid

Sidefejlsfrekvens

- Lad t være tiden siden sidste sidefejl. De er $F = \frac{1}{t}$ den øjeblikkelige sidefejlsfrekvens
  - Er F større end $F_{\max}$ tilføjes en ramme
  - Hvis F er mindre end $F_{\min}$ fjernes alle rammer der ikke er tilgået siden sidste sidefejl
- Problem: ved mange lokalitetsskift tilføjes nye rammer men gamle sider fjernes ikke umiddelbart
- Der er andre alternativer

Thrashing

- Thrashing: når processoren bruger det meste af tiden på at håndtere page faults fremfor at udføre "almindelige" instruktioner
- Hvis $\sum |W_i(t, \Delta)| \leq m$, hvor m er mængden af hukommelse, kan thrashing undgås
- Ellers må processer suspenderes

Opsummering og næste gang

- Simpel lagerallokering
- Paging (segmentering)
- Sidetabel
- Demand paging og sideerstatningsalgoritmer
- Rammeallokering
- Næste gang: The Great OS Shoot-Out