IF3055 - Manajemen Memori
Virtual Memory

Henny Y. Zubir
STEI - ITB

Ikhtisar

- Demand Paging
- Demand Segmentation

Latar Belakang

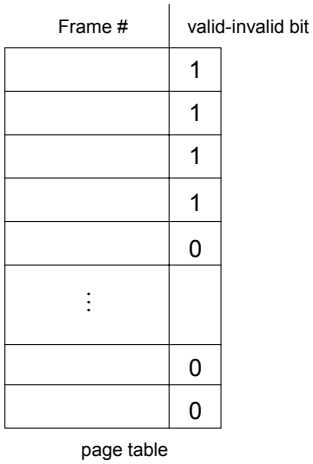
- Virtual memory – pemisahan memori lojik dari memori fisik
  - Hanya sebagian dari program yg perlu berada di memori untuk eksekusi
  - Ruang alamat lojik bisa lebih besar dari ruang alamat fisik
  - Perlu mekanisme untuk swap in/out pages
- Virtual memori dapat diimplementasikan melalui:
  - Demand paging
  - Demand segmentation

Demand Paging

- Page dibawa ke memori hanya jika diperlukan
  - Lebih sedikit I/O yg diperlukan
  - Lebih sedikit memori yg diperlukan
  - Respon lebih cepat
  - Lebih banyak user
- Jika page diperlukan → lakukan pengacuan
  - Jika acuan tidak valid ⇒ abort
  - Jika tidak di memori ⇒ bawa ke memori

Demand Paging: Bit Valid - Invalid

- Tiap entri page table entry memiliki bit valid–invalid
  (1 ⇒ di-memory, 0 ⇒ tidak di memory)
- Bit valid–invalid diinisialisasi 0 utk semua entri
- Pd saat translasi alamat, jika bit valid–invalid pada entri page table = 0 ⇒ page fault



Demand Paging: Page Fault (1)

- Jika ada pengacuan ke suatu page, acuan pertama akan menyebabkan trap ke OS → page fault
- OS mencari di tabel lain utk memutuskan:
  - Pengacuan yg tidak valid ⇒ abort.
  - Tidak berada di memori
- Cari frame kosong
- Swap page ke frame
- Reset tables, bit validation = 1.

Demand Paging: Page Fault (2)

- Bagaimana jika page tidak ada frame kosong?
  - **Penggantian page** – cari page di memori yg dapat di-swap out
    - algoritma
    - performansi – algoritma dengan page fault minimal
  - Page yg sama bisa dibawa ke memori beberapa kali

Demand Paging: Performansi (1)

- Page Fault Rate $0 \leq p \leq 1.0$
  - Jika $p = 0$, tidak ada page fault
  - Jika $p = 1$, tiap acuan ke memori menyebabkan page fault
- **Effective Access Time (EAT)**

  EAT = (1 – p) x akses memory
  + p (overhead page fault
  + [swap page out]
  + swap page in
  + restart overhead)

Demand Paging: Performansi (2)

- Contoh:
  - Waktu akses memory = 1 ms
  - 50% page yg digantikan telah dimodifikasi, dan karena itu perlu di-swap out
  - Waktu swap page = 10 ms = 10,000 ms

EAT = (1 – p) x 1 + p (15000)

1 + 15000P (dalam ms)

Demand Paging: Penggantian Page

- Mencegah over alokasi memori dengan melakukan swap out page yang ada di memori
- Menggunakan **bit modify (dirty)** untuk mengurangi overhead untuk mentransfer page – hanya page yg dimodifikasi yg ditulis ke disk
- Penggantian page menuntaskan pemisahan antara memori lojik dan memori fisik – virtual memory yg besar dapat disediakan pada memori fisik yg lebih kecil

Algoritma Penggantian Page

- Tujuan: minimum page-fault rate
- Evaluasi algoritma dgn menjalankan pd string acuan memori (reference string) dan menghitung banyaknya page fault yg terjadi pd string tsb

Algoritma Penggantian Page: FIFO

Contoh reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

- 3 frame

1	1	4	5	
2	2	1	3	9 page faults
3	3	2	4	

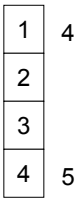
- 4 frames

1	1	5	4	
2	2	1	5	10 page faults
3	3	2		
4	4	3		

- Penggantian FIFO – Belady's Anomaly
  - Lebih banyak frames ⇒ lebih sedikit page faults

Algoritma Penggantian Page: Optimal

- Ganti page yg tidak akan digunakan untuk jangka waktu yg paling lama
- Contoh 4 frames: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5



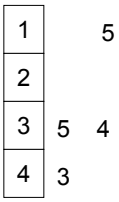
6 page faults

Bagaimana mengetahuinya?

- Digunakan untuk mengukur performansi algoritma lainnya

Algoritma Penggantian Page: LRU (1)

- Reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5



- Implementasi counter
  - Tiap entri page memiliki counter; tiap kali page diacu, salin clock ke counter
  - Jika suatu page perlu giganti, lihat di counter utk menentukan page mana yg akan diganti

Algoritma Penggantian Page: LRU (2)

- Implementasi stack – simpan stack dari page# dalam bentuk link ganda:
  - Page yg diacu:
    - Pindahkan ke atas
    - Memerlukan 6 pointer utk dirubah
  - Penggantian page tidak memerlukan pencarian

Algoritma Penggantian Page: Aproksimasi LRU

- Reference bit
  - Tiap page memiliki satu bit, diinisialisasi 0
  - Jika page diacu, bit diset 1
  - Gantikan page dgn reference bit 0 (jika ada)
- Second chance
  - Perlu reference bit
  - Penggantian clock
  - Jika page digantikan (dlm urutan clock) memiliki reference bit = 1, maka:
    - set reference bit 0
    - biarkan page di memori
    - ganti next page (dlm urutan clock), dengan aturan yg sama

Algoritma Penggantian Page: Counting

- Counter berisi banyaknya acuan yg telah dibuat utk tiap page
- **Algoritma LFU**: ganti page yg memiliki hitungan paling kecil
- **Algoritma MFU**: berdasarkan asumsi bahwa page dgn hitungan terkecil mungkin baru saja dibawa ke memori dan baru digunakan

Alokasi Frame

- Tiap proses memerlukan banyak page minimum
- Contoh: IBM 370 – 6 page utk menangani instruksi SS MOVE
  - instruksi terdiri dari 6 byte, bisa mencakup 2 page
  - 2 page utk menangani **from**
  - 2 page utk menangani **to**
- Dua skema alokasi utama
  - Alokasi fixed
  - Alokasi prioritas

Alokasi Fixed

- Alokasi yg sama – contoh, jika ada 100 frames dan 5 proses, tiap proses memperoleh 20 frames
- Alokasi proporsional – alokasikan berdasarkan ukuran proses

$s_i = \text{size of process } p_i$

$S = \sum s_i$

$m = \text{total number of frames}$

$a_i = \text{allocation for } p_i = \frac{s_i}{S} \times m$

$m = 64$

$s_i = 10$

$s_2 = 127$

$a_1 = \frac{10}{137} \times 64 \approx 5$

$a_2 = \frac{127}{137} \times 64 \approx 59$

Alokasi prioritas

- Skema alokasi proporsional berdasarkan prioritas, bukan ukuran
- Jika proses P_i membangkitkan page fault,
  - pilih utk penggantian salah satu dari frame-nya
  - pilih utk penggantian frame dari proses dgn prioritas lebih rendah

Alokasi Global vs Alokasi Lokal

- **Penggantian global** – proses memilih frame penggantian dari set semua frame; satu proses dapat mengambil frame dari yang lain
- **Penggantian local** – tiap proses memilih dari set frame yang dialokasikan untuknya

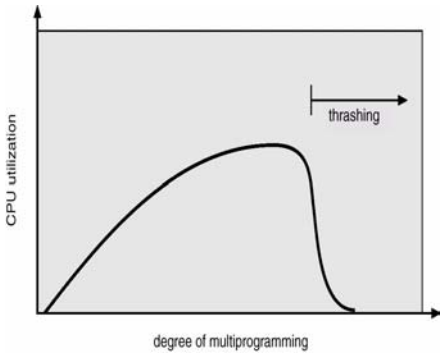
Thrashing

- Jika proses tidak memiliki page yg "cukup", angka page fault menjadi sangat tinggi.

  Hal ini mengakibatkan:
  - Utilisasi CPU rendah
  - OS menilai perlu meningkatkan level multiprogramming
  - Proses lain ditambahkan ke sistem
- Thrashing ≡ proses sibuk melakukan swapping pages in dan out

Thrashing: Diagram

- Mengapa paging bisa diandalkan?
  Model lokalitas
  - Proses migrasi dari satu lokalitas ke yang lain
  - Lokalitas mungkin tumpang tindih
- Mengapa thrashing terjadi?
  - Σ ukuran lokalitas > ukuran total memory



Working Set Model

- Δ ≡ working-set window ≡ banyak acuan page yg fixed
  Contoh: 10,000 instruksi
- WSS_i (working set dari proses P_i) =
  total banyaknya yg diacu sejak Δ waktu terakhir (bervariasi)
  - Jika Δ terlalu kecil tidak akan mencakup keseluruhan lokalitas
  - Jika Δ terlalu besar akan mencakup beberapa lokalitas
  - Jika Δ = ∞ ⇒ mencakup keseluruhan program
- $D = \Sigma\ WSS_i \equiv$ total demand frames
- Jika $D > m \Rightarrow$ Thrashing, suspend salah satu proses

Mengelola Working Set

- Perkiraan dengan timer interval + bit reference
- Contoh: Δ = 10,000
  - Timer interrupts setiap 5000 unit waktu
  - Simpan 2 bits di memori untuk tiap page
  - Jika terjadi timer interrupts, salin dan set nilai semua bit reference ke 0
  - Jika salah satu bit di memori = 1 $\Rightarrow$ page ada di working set
- Mengapa cara ini tidak terlalu akurat?
- Perbaikan = 10 bit dan interrupt setiap 1000 unit waktu

Demand Segmentation

- Digunakan jika dukungan hardware utk mengimplementasikan demand paging tidak memadai
- OS/2 mengalokasikan memori dalam segment, yg dikelola melalui segment descriptor
- Segment descriptor berisi bit valid untuk menunjukkan apakah segment sedang berada di memori
  - Jika segment berada di memori utama, teruskan akses
  - Jika tidak berada di memori, terjadi segment fault