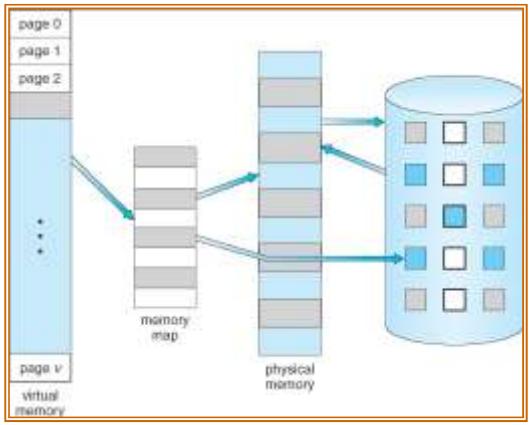
PERTEMUAN VI
VIRTUAL MEMORI

Memori virtual adalah suatu teknik yang memisahkan antara memori logis dan memori fisiknya. Teknik ini menyembunyikan aspek-aspek fisik memori dari pengguna dengan menjadikan memori sebagai lokasi alamat virtual berupa byte yang tidak terbatas dan menaruh beberapa bagian dari memori virtual yang berada di memori logis.

Konsep memori virtual dikemukakan pertama kali oleh John Fotheringham pada tahun 1961 dengan menggunakan dynamic storage allocation pada sistem komputer atlas di Universitas Manchester. Sedangkan istilah memori virtual dipopulerkan oleh Peter J. Denning yang mengambil istilah 'virtual' dari dunia optik.



Gambar 1 Memori virtual lebih besar ukurannya dari memori fisik

Setiap program yang dijalankan harus berada di memori. Memori merupakan suatu tempat penyimpanan utama (*primary storage*) yang bersifat sementara (*volatile*). Ukuran memori yang terbatas menimbulkan masalah bagaimana menempatkan program yang berukuran lebih besar dari ukuran memori fisik dan masalah penerapan multiprogramming yang membutuhkan tempat lebih besar di memori. Dengan pengaturan oleh sistem operasi dan didukung perangkat keras, memori virtual dapat mengatasi masalah kebutuhan memori tersebut.

Memori virtual melakukan pemisahan dengan menaruh memori logis ke disk sekunder dan hanya membawa halaman yang diperlukan ke memori utama. Teknik ini menjadikan seolah-olah ukuran memori fisik yang dimiliki lebih besar dari yang sebenarnya dengan menempatkan keseluruhan program di disk sekunder dan membawa halaman-halaman yang diperlukan ke memori fisik. Jadi jika proses yang sedang berjalan membutuhkan instruksi atau data yang terdapat pada suatu halaman tertentu maka halaman tersebut akan dicari di memori utama. Jika halaman yang diinginkan tidak ada maka akan dicari di disk. Ide ini seperti menjadikan memori sebagai cache untuk disk.

Beberapa keuntungan penggunaan memori virtual adalah sebagai berikut:

- Berkurangnya proses I/O yang dibutuhkan (lalu lintas I/O menjadi rendah). Misalnya untuk program butuh membaca dari disk dan memasukkan dalam memory setiap kali diakses.
- Ruang menjadi lebih leluasa karena berkurangnya memori fisik yang digunakan. Contoh, untuk program 10 MB tidak seluruh bagian dimasukkan dalam memori fisik. Pesan-pesan error hanya dimasukkan jika terjadi error.
- Meningkatnya respon, karena menurunnya beban I/O dan memori.
- Bertambahnya jumlah pengguna yang dapat dilayani. Ruang memori yang masih tersedia luas memungkinkan komputer untuk menerima lebih banyak permintaan dari pengguna.

Gagasan utama dari memori virtual adalah ukuran gabungan program, data dan stack melampaui jumlah memori fisik yang tersedia. Sistem operasi menyimpan bagian-bagian proses yang sedang

digunakan di memori fisik (memori utama) dan sisanya diletakkan di disk. Begitu bagian yang berada di disk diperlukan, maka bagian di memori yang tidak diperlukan akan dikeluarkan dari memori fisik (swap-out) dan diganti (swap-in) oleh bagian disk yang diperlukan itu.

Memori virtual diimplementasikan dalam sistem multiprogramming. Misalnya: 10 program dengan ukuran 2 Mb dapat berjalan di memori berkapasitas 4 Mb. Tiap program dialokasikan 256 Kbyte dan bagian-bagian proses swap in) masuk ke dalam memori fisik begitu diperlukan dan akan keluar (swap out) jika sedang tidak diperlukan. Dengan demikian, sistem multiprogramming menjadi lebih efisien.

Prinsip dari memori virtual yang perlu diingat adalah bahwa "Kecepatan maksimum eksekusi proses di memori virtual dapat sama, tetapi tidak pernah melampaui kecepatan eksekusi proses yang sama di sistem yang tidak menggunakan memori virtual".

Memori virtual dapat diimplementasikan dengan dua cara:

1. Demand paging. Menerapkan konsep pemberian halaman pada proses.
2. Demand segmentation. Lebih kompleks diterapkan karena ukuran segmen yang bervariasi. Demand segmentation tidak akan dijelaskan pada pembahasan ini.

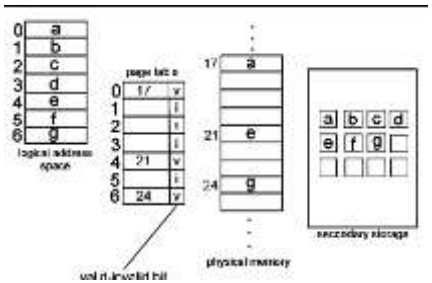
Demand Paging

Demand paging adalah salah satu implementasi dari memori virtual yang paling umum digunakan. Demand paging pada prinsipnya hampir sama dengan permintaan halaman (paging) hanya saja halaman (page) tidak akan dibawa ke ke dalam memori fisik sampai ia benar-benar diperlukan. Untuk itu diperlukan bantuan perangkat keras untuk mengetahui lokasi dari halaman saat ia diperlukan.

Karena demand paging merupakan implementasi dari memori virtual, maka keuntungannya sama dengan keuntungan memori virtual, yaitu:

- Sedikit I/O yang dibutuhkan.
- Sedikit memori yang dibutuhkan.
- Respon yang lebih cepat.
- Dapat melayani lebih banyak pengguna.

Ada tiga kemungkinan kasus yang dapat terjadi pada saat dilakukan pengecekan pada halaman yang dibutuhkan, yaitu: halaman ada dan sudah berada di memori-statusnya valid ("1"); halaman ada tetapi masih berada di disk atau belum berada di memori (harus menunggu sampai dimasukkan)-statusnya tidak valid ("0"). Halaman tidak ada, baik di memori maupun di disk (invalid reference).



Gambar 2 Tabel halaman untuk skema bit valid dan tidak valid

Pengaturan bit dilakukan dengan:
Bit=1 berarti halaman berada di memori. Bit=0
berarti halaman tidak berada di memori.
Apabila ternyata hasil dari translasi, bit halaman bernilai 0, berarti kesalahan halaman terjadi. Kesalahan halaman adalah interupsi yang terjadi ketika halaman yang diminta tidak berada di memori utama. Proses yang sedang berjalan akan mengakses tabel halaman untuk mendapatkan referensi halaman yang diinginkan. Kesalahan halaman dapat diketahui dari penggunaan skema bit valid-tidak valid. Bagian inilah yang menandakan terjadinya suatu permintaan halaman (demand paging). Jika

proses mencoba mengakses halaman dengan bit yang diset tidak valid maka akan terjadi kesalahan halaman. Proses akan terhenti, sementara halaman yang diminta dicari di disk.

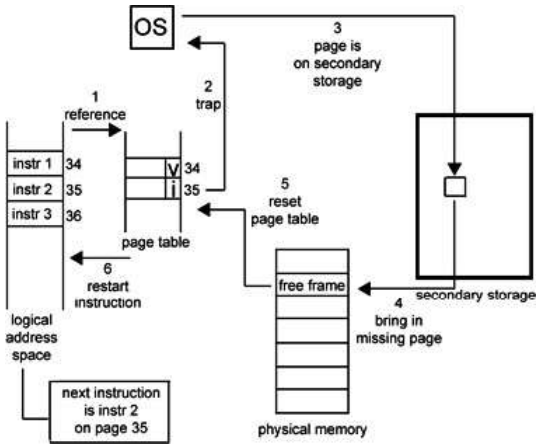
Penanganan Kesalahan Halaman

Penanganan kesalahan halaman dapat dituliskan sebagai berikut:

- CPU mengambil instruksi dari memori untuk dijalankan. Lakukan pengambilan instruksi dari halaman pada memori dengan mengakses tabel halaman. Pada tabel halaman bit terset tidak valid.
- Terjadi interupsi kesalahan halaman, maka interupsi itu menyebabkan trap pada sistem operasi.
- Jika referensi alamat yang diberikan ke sistem operasi ilegal atau dengan kata lain halaman yang ingin diakses tidak ada maka proses akan dihentikan. Jika referensi legal maka halaman yang diinginkan diambil dari disk.
- Halaman yang diinginkan dibawa ke memori fisik.
- Mengatur ulang tabel halaman sesuai dengan kondisi yang baru. Jika tidak terdapat ruang di memori fisik untuk menaruh halaman yang baru maka dilakukan penggantian halaman dengan memilih salah satu halaman. Penggantian halaman dilakukan menurut algoritma tertentu yang akan dibahas pada bab selanjutnya. Jika halaman yang digantikan tersebut sudah dimodifikasi oleh proses maka halaman tersebut harus ditulis kembali ke disk.
- Setelah halaman yang diinginkan sudah dibawa ke memori fisik maka proses dapat diulang.

Kesalahan halaman menyebabkan urutan kejadian berikut:

1. Ditangkap oleh Sistem Operasi.
2. Menyimpan register pengguna dan proses.
3. Tetapkan bahwa interupsi merupakan kesalahan halaman.
4. Periksa bahwa referensi halaman adalah legal dan tentukan lokasi halaman pada disk.
5. Kembangkan pembacaan disk ke frame kosong.
6. Selama menunggu, alokasikan CPU ke pengguna lain dengan menggunakan penjadwalan CPU.
7. Terjadi interupsi dari disk bahwa I/O selesai.
8. Simpan register dan status proses untuk pengguna yang lain.
9. Tentukan bahwa interupsi berasal dari disk.
10. Betulkan tabel halaman dan tabel yang lain bahwa halaman telah berada di memori.
11. Tunggu CPU untuk untuk dialokasikan ke proses tersebut
12. Kembalikan register pengguna, status proses, tabel halaman, dan meneruskan instruksi interupsi.



Gambar 3 Gambaran pada saat penanganan kesalahan halaman

Pada berbagai kasus, ada tiga komponen yang kita hadapi pada saat melayani kesalahan halaman:

- Melayani interupsi kesalahan halaman
- Membaca halaman
- Mengulang kembali proses

Kelebihan/Kekurangan

Manajemen memori dengan permintaan halaman (demand paging) memiliki kelebihan yang sama dengan manajemen memori dengan pemberian halaman, antara lain menghilangkan masalah fragmentasi eksternal sehingga tidak diperlukan pemadatan (*compaction*). Selain itu permintaan halaman memiliki kelebihan yang lain, yaitu:

a. Memori virtual yang besar. Memori logis tidak lagi terbatas pada ukuran memori fisik. Hal ini berarti bahwa besar suatu program tidak akan terbatas hanya pada ukuran memori fisik tersedia.
b. Penggunaan memori yang lebih efisien. Bagian program yang dibawa ke memori fisik hanyalah bagian program yang dibutuhkan sementara bagian lain yang jarang digunakan tidak akan dibawa.
c. Meningkatkan derajat multiprogamming. Derajat multiprogramming menunjukkan banyaknya proses yang berada di memori fisik. Dengan penggunaan permintaan halaman maka ukuran suatu program di memori akan lebih kecil mengingat bahwa hanya bagian program yang diperlukan saja yang akan dibawa ke memori fisik. Penggunaan memori yang lebih kecil oleh sebuah proses memberi sisa ruang memori fisik yang lebih besar sehingga lebih banyak proses yang bisa berada di memori fisik. Hal ini berpengaruh pada utilisasi CPU dan throughput (banyaknya proses yang dapat diselesaikan dalam satu satuan waktu) yang lebih besar.
d. Penggunaan I/O yang lebih sedikit. Hal ini dapat terjadi karena permintaan halaman hanya membawa bagian yang diperlukan dari suatu program. Penggunaan I/O pada permintaan halaman lebih sedikit dibandingkan dengan manajemen memori lain yang membawa seluruh memori logis sebuah program ke memori fisik.

Permintaan halaman juga memiliki beberapa kekurangan, antara lain:

a. Processor overhead. Interupsi kesalahan halaman memberikan kerja tambahan kepada CPU untuk mengambil halaman yang tidak berada di memori fisik pada saat diperlukan.
b. Thrashing. Suatu kondisi yang terjadi akibat kesalahan halaman yang melewati batas normal. Akibat dari thrashing adalah CPU lebih banyak mengurusi kesalahan halaman daripada menangani proses itu sendiri. Hal ini dapat menurunkan kinerja dari CPU.

Kinerja Demand paging

Salah satu hal yang menjadi pertimbangan dalam penggunaan permintaan halaman adalah waktu akses memori menjadi lebih lambat akibat perlunya penanganan kesalahan halaman.

Halaman Fault Time

Lamanya waktu untuk mengatasi kesalahan halaman disebut dengan halaman fault time. Ada tiga faktor utama yang mempengaruhi halaman fault time ini, yaitu:

1. Melayani interupsi dari kesalahan halaman. Aktivitas yang dilakukan dalam melayani kesalahan halaman ini, yaitu:
   a. Memberitahu sistem operasi saat terjadinya kesalahan halaman.
   b. Menyimpan status dari proses bersangkutan.
   c. Memeriksa apakah referensi halaman yang diberikan legal atau tidak. Bila referensi yang diberikan legal maka dicari lokasi dari halaman tersebut di disk.

2. Pembacaan halaman. Aktivitas yang terjadi dalam pembacaan halaman ini, yaitu:
   a. Menunggu dalam antrian sampai mendapatkan giliran untuk membaca.
   b. Menunggu disk untuk membaca lokasi yang diminta. Disk melakukan kerja mekanis untuk membaca data sehingga lebih lambat dari memori.
   c. Mengirim halaman yang diminta ke memori fisik.

3. Pengulangan instruksi. Aktivitas yang terjadi untuk mengulangi instruksi ini, yaitu:
   a. Interupsi proses yang sedang berjalan untuk menandakan bahwa proses yang sebelumnya terhenti akibat kesalahan halaman telah selesai dalam membaca halaman yang diminta.
   b. Menyimpan status dari proses yang sedang berjalan.
   c. Membetulkan tabel halaman untuk menunjukkan bahwa halaman yang ingin dibaca sudah ada di memori fisik.
   d. Mengambil kembali status proses bersangkutan untuk selanjutnya dijalankan di CPU.

Effective Access Time

Untuk mengetahui kinerja permintaan halaman dapat dilakukan dengan menghitung effective access time-nya.
Effective Access Time (EAT) = (1-p) x ma + p x halaman fault time

```
- p   = kemungkinan terjadi halaman fault (0 < p < 1)
- ma = memory access time (10 - 200 ns)
```

Jika p = 0 maka tidak ada kesalahan halaman, sehingga EAT = memory access time.
Jika p = 1 maka semua pengaksesan mengalami kesalahan halaman.

Untuk menghitung EAT, kita harus mengetahui waktu yang dibutuhkan untuk melayani kesalahan halaman.

Contoh:
Diketahui waktu pengaksesan memori (ma) = 100 ns, waktu kesalahan halaman (halaman fault time)=20 ms. Berapakah Effective Access Time-nya?

```
EAT    = (1-p) x ma + p x halaman fault time
       = (1-p) x 100 + p x 20.000.000
       = 100 - 100p + 20.000.000p
       = 100 + 19.999.900p nanosecond
```

Pada permintaan halaman diusahakan agar kemungkinan terjadinya halaman fault rendah karena bila EAT-nya meningkat, maka proses akan berjalan lebih lambat.

Persyaratan Perangkat Keras

Locality of References
Jika terjadi banyak kesalahan halaman maka efisiensi sistem akan menurun. Oleh karena itu, kemungkinan terjadinya kesalahan halaman harus dikurangi atau dibatasi dengan memakai prinsip lokalitas ini. Prinsip lokalitas ini dibagi menjadi 2 bagian:

- temporal. Lokasi yang sekarang ditunjuk kemungkinan akan ditunjuk lagi.
- spatial. Kemungkinan lokasi yang dekat dengan lokasi yang sedang ditunjuk akan ditunjuk juga.

Pure Demand Paging
Penjalanan (running) sebuah program dimulai dengan membawa hanya satu halaman awal ke main memori. Setelah itu, setiap kali terjadi permintaan halaman, halaman tersebut baru akan dimasukkan ke dalam memori. Halaman akan dimasukkan ke dalam memori hanya bila diperlukan.

Multiple Page Fault
Kesalahan halaman yang terjadi karena satu instruksi memerlukan pengaksesan beberapa halaman yang tidak ada di memori utama. Kejadian seperti ini dapat mengurangi kinerja dari program. Pemberian nomor halaman melibatkan dukungan perangkat keras, sehingga ada persyaratan perangkat keras yang harus dipenuhi. Perangkat-perangkat keras tersebut sama dengan yang digunakan untuk paging dan swapping, yaitu:

- Tabel halaman "bit valid-tidak valid"
- Valid ("1") artinya halaman sudah berada di memori
- Tidak valid ("0") artinya halaman masih berada di disk.
- Memori sekunder, digunakan untuk menyimpan proses yang belum berada di memori.

Lebih lanjut, sebagai konsekuensi dari persyaratan ini, akan diperlukan pula perangkat lunak yang dapat mendukung terciptanya pemberian nomor halaman.

Restart Instruction
Salah satu penanganan jika terjadi kesalahan halaman adalah kebutuhan akan pengulangan instruksi. Penanganan pengulangam instruksi berbeda-beda tergantung pada kemungkinan terjadinya kesalahan halaman dan kompleksitas instruksi.

1. Jika kesalahan halaman terjadi saat pengambilan instruksi maka pengulangan proses dilakukan dengan mengambil instruksi itu lagi.

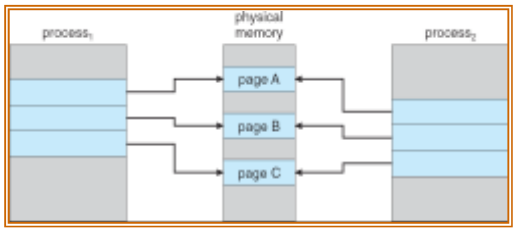
2. Jika kesalahan halaman terjadi saat pengambilan operan maka pengulangan proses dilakukan dengan mengambil instruksi itu lagi, men-dekode-kan, baru mengambil operand.
3. Pengulangan proses ketika instruksi memodifikasi beberapa lokasi yang berbeda, misalnya pada instruksi move character dari blok source ke blok destination dimana kedua blok tersebut tumpang tindih (overlapping) dan kesalahan halaman terjadi saat sebagian instruksi tersebut sudah dijalankan, tidak dapat dilakukan secara langsung karena ada bagian dari blok source yang telah termodifikasi. Ada dua solusi untuk mengatasi hal ini, yaitu:
   a. Komputasi kode mikro dan berusaha untuk mengakses kedua ujung dari blok, agar tidak ada modifikasi halaman yang sempat terjadi.
   b. Menggunakan register sementara (temporary register) untuk menyimpan nilai yang berada pada lokasi yang "overwritten" sehingga bila terjadi kesalahan halaman semua nilai lama dapat ditulis kembali ke memori dan pengulangan dapat dilakukan.
   c. Pengulangan proses ketika instruksi menggunakan mode pengalamatan spesial seperti autoincrement dan autodacrement, misalnya instruksi MOV (R2)+,-(R3) yang menyalin isi lokasi yang ditunjuk register R2 ke lokasi yang ditunjuk register R3. Misalkan R2=10 dan R3=20 maka isi alamat memori 10 akan disalin ke alamat 19. Apabila terjadi kesalahan halaman saat penyimpanan ke lokasi yang ditunjuk R3 dan proses langsung terulang maka penyalinan akan dilakukan dari lokasi 11 ke lokasi 18. Solusinya yaitu dengan memakai register status khusus yang akan menyimpan angka pada register dan jumlah perubahan (modifikasi) yang dialami oleh register sehingga register dapat diubah kembali ke nilai lama dan sistem operasi dapat mengulang sebagian proses instruksi yang telah dilakukan.

Creation Process

Proses berbagi pakai ini adalah proses berbagi pakai halaman memori virtual. Karena setiap proses membutuhkan halaman tersendiri maka akan dibutuhkan teknik untuk mengelola halaman dan pembuatannya. Teknik untuk mengoptimasi pembuatan dan penggunaan halaman proses adalah dengan Copy-On-Write dan Memory-Mapped-File.

Copy-On-Write (Cow)

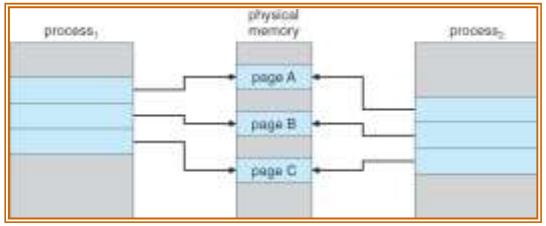
Dengan memanggil sistem pemanggilan fork(), sistem operasi akan membuat proses anak sebagai salinan dari proses induk. Sistem pemanggilan fork() bekerja dengan membuat salinan alamat proses induk untuk proses anak, lalu membuat salinan halaman milik proses induk tersebut.Tapi, karena setelah pembuatan proses anak selesai, proses anak langsung memanggil sistem pemanggilan exec() yang menyalin alamat proses induk yang kemungkinan tidak dibutuhkan.



Gambar 4 Sebelum proses 1 memodifikasi page C

Oleh karenanya, lebih baik menggunakan teknik lain dalam pembuatan proses yang disebut sistem copy-on-write. Teknik ini bekerja dengan memperbolehkan proses anak untuk menginisialisasi penggunaan halaman yang sama secara bersamaan. halaman yang digunakan bersamaan itu, disebut dengan "halaman copy-on-write", yang berarti jika salah satu dari proses anak atau proses induk melakukan penulisan pada halaman tersebut, maka akan dibuat juga sebuah salinan dari halaman itu.

Sebagai contoh, sebuah proses anak hendak memodifikasi sebuah halaman yang berisi sebagian dari stack. Sistem operasi akan mengenali hal ini sebagai copy-on-write, lalu akan membuat salinan dari halaman ini dan memetakannya ke alamat memori dari proses anak, sehingga proses anak akan memodifikasi halaman salinan tersebut, dan bukan halaman milik proses induk. Dengan teknik copy- on-write ini, halaman yang akan disalin adalah halaman yang dimodifikasi oleh proses anak atau proses induk. Halaman-halaman yang tidak dimodifikasi akan bisa dibagi untuk proses anak dan proses induk.



Gambar 5 Sesudah proses 1 memodifikasi page C

Saat suatu halaman akan disalin menggunakan teknik copy-on-write, digunakan teknik zero-fill-on-demand untuk mengalokasikan halaman kosong sebagai tempat meletakkan hasil duplikat. Halaman kosong tersebut dialokasikan saat stack atau heap suatu proses akan diperbesar atau untuk mengatur halaman copy-on-write. Halaman Zero-fill-on-demand akan dibuat kosong sebelum dialokasikan, yaitu dengan menghapus isi awal dari halaman. Karena itu, dengan copy-on-write, halaman yang sedang disalin akan disalin ke sebuah halaman zero-fill-on. Teknik copy-on-write digunakan oleh beberapa sistem operasi seperti Windows 2000, Linux, dan Solaris2.

Dengan COW, beberapa proses dapat berbagi pakai halaman yang sama, namun jika ada salah satu proses akan menulis atau melakukan modifikasi, maka dibuat halaman baru (sebagai salinan dari halaman copy-on-write). Pada halaman salinan tersebut proses melakukan modifikasi. Halaman yang lama tetap. Halaman Copy-On-Write diberi tanda sebagai "halaman Copy-On-Write" dan bersifat "read only", sedangkan halaman salinan tidak diberi tanda dan bisa dimodifikasi.

Misalkan, halaman C digunakan bersama oleh proses 1 dan proses 2. Ketika proses 1 akan melakukan modifikasi terhadap halaman C, maka sistem operasi membuat halaman baru sebagai salinan dari halaman C yang ditunjuk oleh proses 1. Proses 1 melakukan modifikasi pada halaman yang baru tersebut (ditunjuk oleh proses 1), sedangkan halaman C tetap (ditunjuk oleh proses 2).Untuk menentukan halaman baru sebagai salinan dari halaman Copy-On-Write tersebut Sistem Operasi harus mengetahui letak halaman-halaman yang kosong. Beberapa Sistem Operasi menyediakan pool dari halaman-halaman yang kosong. Selain untuk salinan dari halaman Copy-On- Write, halaman-halaman kosong tersebut disediakan untuk proses yang melakukan penambahan stack atau heap.

Teknik yang biasa digunakan oleh sistem operasi untuk menyediakan halaman tersebut disebut zero-fill-on-demand. Teknik ini dilakukan dengan mengosongkan halaman-halaman sebelum digunakan oleh proses yang baru.

Keuntungan teknik COW

a. Jika tidak ada modifikasi pada halaman maka pembuatan salinan dari halaman tidak akan pernah dilakukan atau jumlah memori fisik yang dialokasikan untuk proses tidak pernah bertambah sampai terjadi penulisan data.
b. Penggunaan memori fisik sangat jarang, memori fisik baru digunakan hanya jika terjadi penyimpanan data.

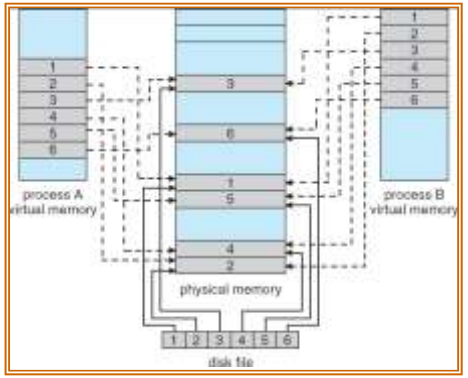
Kekurangan teknik COW adalah bertambahnya kompleksitas pada level kernel, pada saat kernel menulis ke halaman, maka harus dibuat salinan halaman jika halaman tersebut diberi tanda COW.

Memoy Mapped Files

Memory-Mapped Files adalah teknik yang digunakan untuk pengaksesan file, dimana blok berkas dalam disk dipetakan ke halaman memori. File tersebut belum berada dalam memori. Karena pengaksesan berkas dilakukan melalui demand paging, akses kali pertama pada berkas akan menghasilkan halaman fault. Namun sesudah itu sebagian dari halaman berkas akan dibaca dari sistem berkas ke memori fisik. Selanjutnya pembacaan dan penulisan pada berkas dilakukan dalam memori. Hal ini menyederhanakan pengaksesan dan penggunaan berkas daripada pengaksesan langsung ke disk melalui sistem call read() dan write().

Bila terjadi penulisan/perubahan pada bagian berkas dalam memori, perubahan ini tidak langsung dilakukan pada berkas dalam disk. Beberapa sistem operasi melakukan perubahan secara periodik dengan memeriksa apakah ada perubahan pada berkas atau tidak. Ketika berkas ditutup semua perubahan ditulis ke disk dan berkas dihapus dari memori virtual.

Implementasi Memory-Mapped Files (MMF)



Gambar 6 Memory-Mapped Files (MMF)

Pada beberapa sistem operasi, MMF dilakukan dengan menggunakan sistem call khusus sementara sistem call biasa digunakan untuk berkas I/O yang lainya. Pada beberapa sistem lain, walaupun sistem call khusus untuk MMF ada, semua berkas akan dipetakan meskipun file tersebut tidak ditandai sebagai berkas MMF. Pada Solaris misalnya, sistem ini memiliki sistem call mmap() untuk mengaktifkan MMF. Namun Solaris tetap akan memetakan file yang diakses menggunakan sistem call open(), read() dan write().

MMF juga dapat memetakan banyak proses pada berkas yang sama secara bersamaan sehingga proses-proses tersebut dapat saling berbagi data. Pada multiproses, perubahan yang dibuat oleh salah satu proses dapat dilihat oleh proses lainnya. MMF juga mendukung teknik COW. Jika ditambahkan teknik COW, halaman yang dipetakan berada dalam mode read only dan apabila ada proses yang merubah isi berkas maka proses tersebut akan diberi salinan halaman yang akan dirubahnya sehingga perubahan yang terjadi tidak dapat dilihat oleh proses lain.

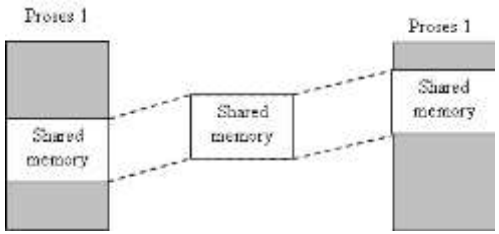
Dalam beberapa hal, sharing memori pada MMF sama dengan sharing memori yang telah dijelaskan pada bab sebelumnya. Unix dan Linux menggunakan mekanisme yang berbeda untuk MMF dan sharing memori, untuk MMF digunakan sistem call mmap(), sedangkan untuk sharing memori digunakan sistem call shmget() dan shmat(). Pada Windows NT, 2000, dan XP penerapan sharing memori dilakukan dengan mekanisme MMF.

Share Memory pada Win32

Untuk melakukan sharing memori menggunakan MMF, pertama kali harus dibuat (create) pemetaan berkas dari disk ke memori. Setelah itu, dibuat view dari berkas tersebut pada alamat virtual proses. Proses kedua (lainnya) dapat membuka dan membuat view dari berkas tersebut dalam ruang alamat virtualnya.

Pemetaan berkas ini merupakan representasi sharing memori, dimana proses dapat berkomunikasi satu sama lain. Misalnya pada proses produsen dan konsumen. Produsen membuat obyek sharing memori menggunakan fasilitas memori mapped-file Win32 API. Kemudian produsen menulis pesan ke dalam memori tersebut. Setelah itu, konsumen dapat membuat pemetaan ke obyek sharing memori tersebut, membukanya, dan membaca pesan yang ditulis oleh produsen.

Untuk membuat memori mapped file, proses harus membuka (open) berkas yang akan dipetakan dengan fungsi CreateFile(). Kemudian proses memetakan berkas dengan fungsi CreateFileMapping(). Setelah berkas dipetakan maka view dari file dapat dimunculkan pada alamat virtual proses dengan menggunakan fungsi MapViewOfFile().



Gambar 7 MMF pada win32 API

Fungsi CreateFileMapping() akan membuat obyek sharing memori yang disebut SharedObject. Konsumen dapat berkomunikasi menggunakan sharing memori dengan membuat pemetaan ke obyek dengan nama yang sama. Sementara fungsi MapViewOfFile() akan memunculkan pointer pada obyek shared-memori, seluruh akses pada bagian memori ini merupakan akses pada memory-mapped file.

Kelebihan Memory-Mapped Files

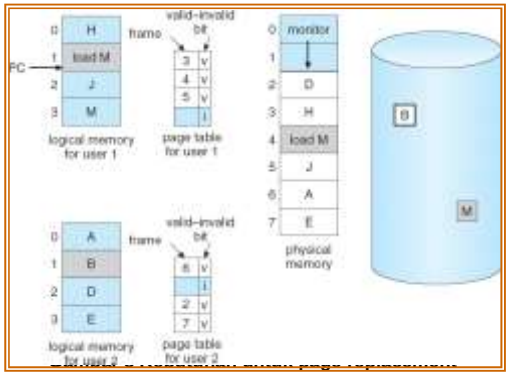
1. Akses pada berkas dilakukan dengan pointer, tidak langsung dengan sistem call read() dan write().
2. Akses dapat dilakukan secara acak.
3. "On demand loading", permintaan I/O baru akan terjadi kalau berkas/bagian berkas yang bersangkutan di akses.
4. Data/bagian berkas yang tak terpakai akan di-swap out.

Kelemahan Memory-Mapped Files

1. Bila terjadi penambahan berkas harus dibuat MMF yang baru.
2. Proses I/O secara asinkron tidak dapat dilakukan, semuanya dilakukan per blok.
3. Terkadang terjadi bug ketika OS menulis ke dalam file.

Page Replacement

Masalah kesalahan halaman pasti akan dialami oleh setiap halaman minimal satu kali. Akan tetapi, sebenarnya sebuah proses yang memiliki N buah halaman hanya akan menggunakan N/2 diantaranya. Kemudian permintaan halaman akan menyimpan I/O yang dibutuhkan untuk mengisi N/2 halaman sisanya. Dengan demikian utilisasi CPU dan throughput dapat ditingkatkan.



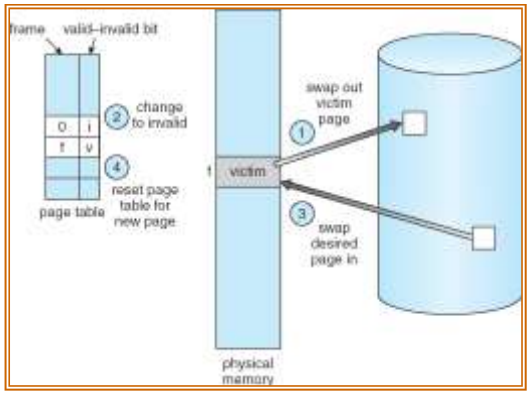
Upaya yang dilakukan oleh permintaan halaman dalam mengatasi kesalahan halaman didasari oleh pemindahan halaman. Sebuah konsep yang akan kita bahas lebih lanjut dalam sub bab ini. Prinsip dalam melakukan pemindahan halaman adalah sebagai berikut: "Jika tidak ada frame yang kosong, cari frame yang tidak sedang digunakan dalam jangka waktu yang lama, lalu kosongkan dengan memindahkan isinya ke dalam ruang pertukaran dan ubah semua tabel halamannya sebagai indikasi bahwa halaman tersebut tidak akan lama berada di dalam memori." Pada kasus pemindahan halaman

di atas, frame kosong yang diperoleh akan digunakan sebagai tempat penyimpanan halaman yang salah.

Rutinitas yang dilakukan dalam pemindahan halaman antara lain:

- Mencari lokasi dari halaman yang diinginkan pada disk.
- Mencari frame yang kosong:
  a. Jika ada, maka gunakan frame tersebut.
  b. Jika tidak ada, maka tentukan frame yang tidak sedang dipakai atau yang tidak akan digunakan dalam jangka waktu lama, lalu kosongkan frame tersebut. Gunakan algoritma pemindahan halaman untuk menentukan frame yang akan dikosongkan. Usahakan agar tidak menggunakan frame yang akan digunakan dalam waktu dekat. Jika terpaksa, maka sebaiknya segera masukkan kembali frame tersebut agar tidak terjadi overhead.
  c. Tulis halaman yang dipilih ke disk, ubah tabel halaman dan tabel frame.
- Membaca halaman yang diinginkan ke dalam frame kosong yang baru.
- Mengulangi proses pengguna dari awal.

Rutinitas di atas belum tentu berhasil. Jika kita tidak dapat menemukan frame yang kosong atau akan dikosongkan, maka sebagai jalan keluarnya kita dapat melakukan pentransferan dua halaman (satu masuk, satu keluar). Cara ini akan menambah waktu pelayanan kesalahan halaman dan waktu akses efektif. Oleh karena itu, perlu digunakan bit tambahan untuk masing-masing halaman dan frame yang diasosiasikan dalam perangkat keras.



Gambar 9 Page Replacement

Sebagai dasar dari permintaan halaman, pemindahan halaman merupakan "jembatan pemisah" antara memori logis dan memori fisik. Mekanisme yang dimilikinya memungkinkan memori virtual berukuran sangat besar dapat disediakan untuk pemrogram dalam bentuk memori fisik yang berukuran lebih kecil.

Dalam permintaan halaman, jika kita memiliki banyak proses dalam memori, kita harus menentukan jumlah frame yang akan dialokasikan ke masing-masing proses. Ketika pemindahan halaman diperlukan, kita harus memilih frame yang akan dipindahkan (dikosongkan). Masalah ini dapat diselesaikan dengan menggunakan algoritma pemindahan halaman.

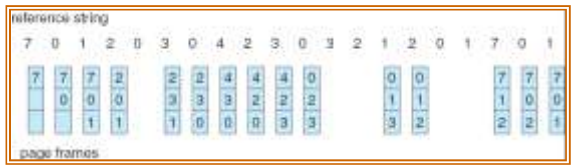
Ada beberapa macam algoritma pemindahan halaman yang dapat digunakan. Algoritma yang terbaik adalah yang memiliki tingkat kesalahan halaman terendah. Selama jumlah frame meningkat, jumlah kesalahan halaman akan menurun. Peningkatan jumlah frame dapat terjadi jika memori fisik diperbesar.

Evaluasi algoritma pemindahan halaman dapat dilakukan dengan menjalankan sejumlah string acuan di memori dan menghitung jumlah kesalahan halaman yang terjadi. Sebagai contoh, suatu proses memiliki urutan alamat: 0100, 0432, 0101, 0612, 0102, 0103, 0104, 0101, 0611, 0102, 0103, 0104,

0101, 0610, 0102, 0103, 0104, 0101, 0609, 0102, 0105; per 100 bytes-nya dapat kita turunkan menjadi string acuan: 1, 4, 1, 6, 1, 6, 1, 6, 1, 6, 1.

Pemindahan halaman diimplementasikan dalam algoritma yang bertujuan untuk menghasilkan tingkat kesalahan halaman terendah. Ada beberapa algoritma pemindahan halaman yang berbeda. Pemilihan halaman yang akan diganti dalam penggunaan algoritma-algoritma tersebut bisa dilakukan dengan berbagai cara, seperti dengan memilih secara acak, memilih dengan berdasarkan pada penggunaan, umur halaman, dan lain sebagainya. Pemilihan algoritma yang kurang tepat dapat menyebabkan peningkatan tingkat kesalahan halaman sehingga proses akan berjalan lebih lambat.

Algoritma FIFO (First In First Out)



Gambar 10 FIFO Page Replacement

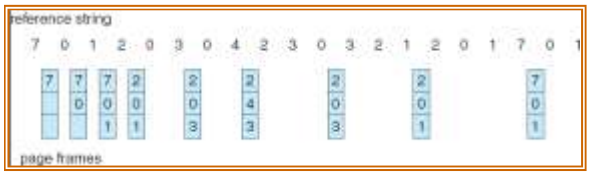
Prinsip yang digunakan dalam algoritma FIFO yaitu menggunakan konsep antrian, halaman yang diganti adalah halaman yang paling lama berada di memori. Algoritma ini adalah algoritma pemindahan halaman yang paling mudah diimplementasikan, akan tetapi paling jarang digunakan dalam keadaan sebenarnya. Biasanya penggunaan algoritma FIFO ini dikombinasikan dengan algoritma lain.

Implementasi algoritma FIFO dilakukan dengan menggunakan antrian untuk menandakan halaman yang sedang berada di dalam memori. Setiap halaman baru yang diakses diletakkan di bagian belakang (ekor) dari antrian. Apabila antrian telah penuh dan ada halaman yang baru diakses maka halaman yang berada di bagian depan (kepala) dari antrian akan diganti.

Kelemahan dari algoritma FIFO adalah kinerjanya yang tidak selalu baik. Hal ini disebabkan karena ada kemungkinan halaman yang baru saja keluar dari memori ternyata dibutuhkan kembali. Di samping itu dalam beberapa kasus, tingkat kesalahan halaman justru bertambah seiring dengan meningkatnya jumlah frame, yang dikenal dengan nama anomali Belady.

Algoritma Optimal

Algoritma optimal pada prinsipnya akan mengganti halaman yang tidak akan digunakan untuk jangka waktu yang paling lama. Kelebihannya antara lain dapat menghindari terjadinya anomali Belady dan juga memiliki tingkat kesalahan halaman yang terendah diantara algoritma-algoritma pemindahan halaman yang lain.



Gambar 11 Optimal Page Replacement

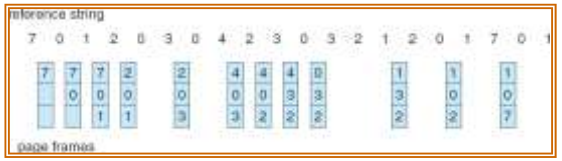
Meski pun tampaknya mudah untuk dijelaskan, tetapi algoritma ini sulit atau hampir tidak mungkin untuk diimplementasikan karena sistem operasi harus dapat mengetahui halaman-halaman mana saja yang akan diakses berikutnya, padahal sistem operasi tidak dapat mengetahui halaman yang muncul di waktu yang akan datang.

Algoritma Least Recenly Used (LRU)

Cara kerja algoritma LRU adalah menggantikan halaman yang sudah tidak digunakan dalam jangka waktu yang paling lama. Pertimbangan algoritma ini yaitu berdasarkan observasi bahwa halaman yang sering diakses kemungkinan besar akan diakses kembali. Sama halnya dengan algoritma optimal, algoritma LRU juga tidak akan mengalami anomali Belady. Namun, sama halnya juga dengan algoritma optimal, algoritma LRU susah untuk diimplementasikan, walaupun sedikit lebih mudah dari algoritma optimal. Pengimplementasian algoritma LRU dapat dilakukan dengan dua cara.

Counter
Cara ini dilakukan dengan menggunakan counter atau logical clock. Setiap halaman memiliki nilai yang pada awalnya diinisialisasi dengan 0. Ketika mengakses ke suatu halaman baru, nilai pada clock di halaman tersebut akan bertambah 1. Untuk melakukan hal itu dibutuhkan extra write ke memori. Halaman yang diganti adalah halaman yang memiliki nilai clock terkecil. Kekurangan dari cara ini adalah hanya sedikit sekali mesin yang mendukung hardware counter.

Stack
Cara ini dilakukan dengan menggunakan stack yang menandakan halaman-halaman yang berada di memori. Setiap kali suatu halaman diakses, akan diletakkan di bagian paling atas stack. Apabila ada halaman yang perlu diganti, maka halaman yang berada di bagian paling bawah stack akan diganti sehingga setiap kali halaman baru diakses tidak perlu mencari kembali halaman yang akan diganti. Dibandingkan pengimplementasian dengan counter, cost untuk mengimplementasikan algoritma LRU dengan menggunakan stack akan lebih mahal karena isi stack harus di-update setiap kali mengakses halaman.



Gambar 12 LRU Page Replacement

Algoritma Perkiraan LRU

Pada dasarnya algoritma perkiraan LRU memiliki prinsip yang sama dengan algoritma LRU, yaitu halaman yang diganti adalah halaman yang tidak digunakan dalam jangka waktu terlama, hanya saja dilakukan modifikasi pada algoritma ini untuk mendapatkan hasil yang lebih baik. Perbedaannya dengan algoritma LRU terletak pada penggunaan bit acuan. Setiap halaman yang berbeda memiliki bit acuan. Pada awalnya bit acuan diinisialisasikan oleh perangkat keras dengan nilai 0. Nilainya akan berubah menjadi 1 bila dilakukan akses ke halaman tersebut. Terdapat beberapa cara untuk mengimplementasikan algoritma ini:

Algoritma NFU (Not Frequently Used)
Pada prinsipnya algoritma LRU dapat diimplementasikan, jika ada mesin yang menyediakan hardware counter atau stack seperti yang digunakan pada LRU. Pendekatan lainnya adalah dengan mengimplementasikannya pada software. Salah satu solusinya disebut algoritma NFU (Not Frequently Used).

Seperti LRU, NFU juga menggunakan counter tapi counter ini diimplementasikan pada software. Setiap halaman memiliki counter, yang diinisiasi dengan nol. Pada setiap clock interval, semua halaman yang pernah diakses pada interval tersebut akan ditambah nilai counter-nya dengan bit R, yang bernilai 0 atau 1. Maka, halaman dengan nilai counter terendah akan diganti.

Kelemahan NFU adalah NFU menyimpan informasi tentang sering- tidaknya sebuah halaman diakses tanpa mempertimbangkan rentang waktu penggunaan. Jadi, halaman yang sering diakses pada pass pertama akan dianggap sebagai halaman yang sangat dibutuhkan pada pass kedua, walaupun halaman tersebut tidak lagi dibutuhkan pada pass kedua karena nilai counter-nya tinggi. Hal ini dapat mengurangi kinerja algoritma ini.

Setiap halaman akan memiliki bit acuan yang terdiri dari 8 bit (byte) sebagai penanda. Pada awalnya semua bit nilainya 0, contohnya: 00000000. Setiap selang beberapa waktu, pencatat waktu melakukan interupsi kepada sistem operasi, kemudian sistem operasi menggeser 1 bit ke kanan dengan bit yang paling kiri adalah nilai dari bit acuan, yaitu bila halaman tersebut diakses nilainya 1 dan bila tidak diakses nilainya 0. Jadi halaman yang selalu digunakan pada setiap periode akan memiliki nilai 11111111. Halaman yang diganti adalah halaman yang memiliki nilai terkecil.

Algoritma Aging
Algoritma Aging adalah turunan dari NFU yang mempertimbangkan rentang waktu penggunaan suatu halaman. Tidak seperti NFU yang hanya menambahkan bit R pada counter, algoritma ini menggeser bit counter ke kanan (dibagi 2) dan menambahkan bit R di paling kiri bit counter. Halaman yang akan diganti adalah halaman yang memiliki nilai counter terendah. Contoh, jika suatu halaman diakses pada interval pertama, tidak diakses dalam dua interval selanjutnya, diakses pada dua interval berikutnya, tidak diakses dalam interval berikutnya, dan seterusnya, maka bit R dari halaman itu: 1, 0, 0, 1, 1, 0, ... Dengan demikian counter-nya akan menjadi: 10000000, 01000000, 00100000,
10010000, 11001000, 01100100, ...

Algoritma ini menjamin bahwa halaman yang paling baru diakses, walaupun tidak sering diakses, memiliki prioritas lebih tinggi dibanding halaman yang sering diakses sebelumnya. Yang perlu diketahui, aging dapat menyimpan informasi pengaksesan halaman sampai 16 atau 32 interval sebelumnya. Hal ini sangat membantu dalam memutuskan halaman mana yang akan diganti. Dengan demikian, aging menawarkan kinerja yang mendekati optimal dengan harga yang cukup rendah.

Algoritma Second-Chance
Algoritma ini adalah modifikasi dari FIFO yang, seperti namanya, memberikan kesempatan kedua bagi suatu halaman untuk tetap berada di dalam memori karena halaman yang sudah lama berada di memori mungkin saja adalah halaman yang sering digunakan dan akan digunakan lagi. Kesempatan kedua itu direalisasikan dengan adanya bit acuan yang di-set untuk suatu halaman.

Penempatan halaman pada antrian sama seperti pada FIFO, halaman yang lebih dulu diakses berada di depan antrian dan yang baru saja diakses berada di belakang antrian. Ketika terjadi kesalahan halaman, algoritma ini tidak langsung mengganti halaman di depan antrian tapi terlebih dahulu memeriksa bit acuannya. Jika bit acuan = 0, halaman tersebut akan langsung diganti. Jika bit acuan = 1, halaman tersebut akan dipindahkan ke akhir antrian dan bit acuannya diubah menjadi 0, kemudian mengulangi proses ini untuk halaman yang sekarang berada di depan antrian.

Algoritma ini dapat terdegenerasi menjadi FIFO ketika semua bit acuan di-set 1. Ketika halaman pertama kembali berada di awal antrian, bit acuan pada semua halaman sudah diubah menjadi 0, sehingga halaman pertama langsung diganti dengan halaman baru.

Proses pemindahan halaman yang diberi kesempatan kedua akan membuat algoritma menjadi tidak efisien karena harus terus memindahkan halaman dalam antrian. Untuk mengatasi masalah ini, digunakan antrian berbentuk lingkaran yang dijelaskan pada algoritma di bawah ini. Pengimplementasian algoritma ini dilakukan dengan menyimpan halaman pada sebuah linked-list dan halaman-halaman tersebut diurutkan berdasarkan waktu ketika halaman tersebut tiba di memori yang berarti menggunakan juga prinsip algoritma FIFO disamping menggunakan algoritma LRU.Apabila nilai bit acuan-nya 0, halaman dapat diganti. Dan apabila nilai bit acuan-nya 1, halaman tidak diganti tetapi bit acuan diubah menjadi 0 dan dilakukan pencarian kembali.

Algoritma Clock
Seperti yang disebutkan di atas, algoritma Clock Halaman (atau Clock saja) menggunakan prinsip Second-Chance tapi dengan antrian yang berbentuk melingkar. Pada antrian ini terdapat pointer yang menunjuk ke halaman yang paling lama berada di antrian. Ketika terjadi kesalahan halaman, halaman yang ditunjuk oleh pointer akan diperiksa bit acuannya seperti pada Second-Chance. Jika bit acuan = 0, halaman tersebut akan langsung diganti. Jika bit acuan = 1, bit acuannya diubah menjadi 0 dan pointer akan bergerak searah jarum jam ke halaman yang berada di sebelahnya.

Penggunaan antrian berbentuk melingkar menyebabkan halaman tidak perlu dipindahkan setiap saat, yang bergerak cukup pointer saja. Meski pun algoritma second-chance sudah cukup baik, namun pada kenyataannya penggunaan algortima tersebut tidak efisien. Algoritma clock adalah penyempurnaan dari algoritma tersebut. Pada prinsipnya kedua algoritma tersebut sama, hanya terletak perbedaan pada pengimplementasiannya saja. Algortima ini menggunakan antrian melingkar

yang berbentuk seperti jam dengan sebuah penunjuk yang akan berjalan melingkar mencari halaman untuk diganti.

Algoritma Counting

Dilakukan dengan menyimpan pencacah dari nomor acuan yang sudah dibuat untuk masing-masing halaman. Penggunaan algoritma ini memiliki kekurangan yaitu lebih mahal. Algoritma Counting dapat dikembangkan menjadi dua skema dibawah ini:

Algoritma Least Frequently Used (LFU)
Halaman yang diganti adalah halaman yang paling sedikit dipakai (nilai pencacah terkecil) dengan alasan bahwa halaman yang digunakan secara aktif akan memiliki nilai acuan yang besar.

Algoritma Most Frequently Used (MFU)
Halaman yang diganti adalah halaman yang paling sering dipakai (nilai pencacah terbesar) dengan alasan bahwa halaman dengan nilai terkecil mungkin baru saja dimasukkan dan baru digunakan.

Algoritma NRU (Not Recently Used)

Dalam algoritma ini terdapat bit acuan dan bit modifikasi yang akan di-update setiap kali mengakses halaman. Ketika terjadi halaman fault, sistem operasi akan memeriksa semua halaman dan membagi halaman-halaman tersebut ke dalam kelas-kelas. Algoritma NRU mudah untuk dimengerti, efisien, dan memiliki kinerja yang cukup baik. Algoritma ini mempertimbangkan dua hal sekaligus, yaitu bit acuan dan bit modifikasi. Ketika terjadi kesalahan halaman, sistem operasi memeriksa semua halaman dan membagi halaman-halaman tersebut ke dalam empat kelas:

a. Kelas 1: Tidak digunakan dan tidak dimodifikasi, Bit terbaik untuk dipindahkan.
b. Kelas 2: Tidak digunakan tapi dimodifikasi, tidak terlalu baik untuk dipindahkan karena halaman ini perlu ditulis sebelum dipindahkan.
c. Kelas 3: Digunakan tapi tidak dimodifikasi, ada kemungkinan halaman ini akan segera digunakan lagi.
d. Kelas 4: Digunakan dan dimodifikasi, halaman ini mungkin akan segera digunakan lagi dan halaman ini perlu ditulis ke disk sebelum dipindahkan.

Halaman yang akan diganti adalah halaman dari kelas yang paling rendah. Jika terdapat lebih dari satu kelas terendah yang sama, maka sistem operasi akan memilih secara acak. Kelebihan algoritma ini adalah mudah untuk dimengerti, efisien, dan memiliki kinerja yang cukup baik.

Algoritma Page Buffering

Pada algoritma ini, sistem menyimpan *pool* dari frame yang kosong dan dijadikan sebagai buffer. Prosedur ini memungkinkan suatu proses mengulang dari awal secepat mungkin, tanpa perlu menunggu halaman yang akan dipindahkan untuk ditulis ke disk karena frame-nya telah ditambahkan ke dalam pool frame kosong. Teknik seperti ini digunakan dalam sistem VAX/VMS.

Kelebihan algoritma ini adalah dapat membuat proses lebih cepat dikembalikan ke status ready queue, namun akan terjadi penurunan kinerja karena sedikit sekali halaman yang digunakan sehingga halaman lain yang tidak digunakan menjadi sia-sia (utilitas sistem rendah).

DISKUSI

Apa fitur dari Copy-On-Write dan dalam keadaan apa fitur ini akan bermanfaat?

Referensi:

[Silberschantz2005] Abraham Silberschantz, Peter Baer Galvin & Greg Gagne. 2005. ***Operating System Concepts.*** Seventh Edition. John Wiley & Son

[MDGR2006] Masyarakat Digital Gotong Royong (MDGR). 2006. ***Pengantar Sistem Operasi Komputer Plus Illustrasi Kernel Linux***. http://bebas.vlsm.org/v06/Kuliah/SistemOperasi/BUKU/. Diakses 31 Agustus 2007