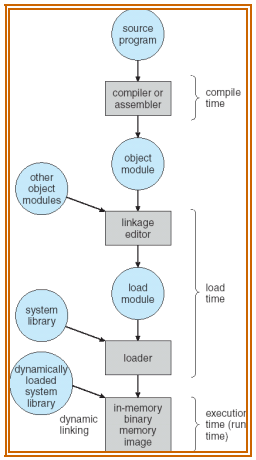
MANAJEMEN MEMORI

Memori merupakan pusat kegiatan pada sebuah komputer, karena setiap proses yang akan dijalankan harus melalui memori terlebih dahulu. CPU mengambil instruksi dari memori sesuai yang ada pada program counter. Instruksi memerlukan proses memasukkan/menyimpan ke alamat di memori. Tugas sistem operasi adalah mengatur peletakan proses pada suatu memori. Algoritma untuk manajemen memori ini bervariasi dari yang menggunakan pendekatan primitif pada mesin sampai dengan pemberian halaman (*page*) dan strategi segmentasi (*segment*). Memori harus dapat digunakan dengan baik, sehingga dapat memuat banyak proses pada suatu waktu.

Address Binding

Umumnya sebuah program ditempatkan dalam disk dalam bentuk berkas biner yang dapat dieksekusi. Sebelum dieksekusi, sebuah program harus ditempatkan di memori terlebih dahulu. Kumpulan proses yang ada pada disk harus menunggu dalam antrian (*input queue*) sebelum dibawa ke memori dan dieksekusi. Prosedur penempatan yang biasa dilakukan adalah dengan memilih salah satu proses yang ada di input queue, kemudian proses tersebut ditempatkan ke memori. Sebelum dieksekusi, program akan melalui beberapa tahap. Dalam setiap tahap alamat sebuah program akan direpresentasikan dengan cara yang berbeda. Alamat di dalam sebuah sumber program biasanya dalam bentuk simbol-simbol.



Gambar 1 Multistep processing dari program user

Sebuah kompilator akan memetakan simbol-simbol ini ke alamat relokasi. Linkage editor akan memetakan alamat relokasi ini menjadi alamat absolut. *Binding* adalah pemetaan dari satu ruang alamat ke alamat yang lain. *Binding* instruksi dan data ke memori dapat dapat terjadi dalam tiga cara yang berbeda:

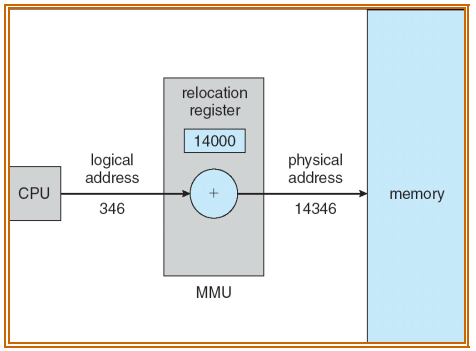
a. ***Compilation Time***. Jika kita tahu dimana proses akan ditempatkan di memori pada saat mengkompilasi, maka alamat absolut dapat dibuat. Kita harus mengkompilasi ulang kode jika lokasi berubah.
b. ***Load Time***. Kita harus membuat kode relokasi jika pada saat mengkompilasi kita tidak mengetahui proses yang akan ditempatkan dalam memori. Pada kasus ini, *binding* harus ditunda sampai *load time*.
c. ***Execution Time***. *Binding* harus ditunda sampai waktu proses berjalan selesai jika pada saat dieksekusi proses dapat dipindah dari satu segmen ke segmen yang lain di dalam memori. Kita butuh perangkat keras khusus untuk melakukan ini.

Pengalamatan Logika dan Fisik

Alamat logika adalah alamat yang dihasilkan oleh CPU, disebut juga alamat virtual. Alamat fisik adalah alamat memori yang sebenarnya. Pada saat waktu kompilasi dan waktu pemanggilan, alamat fisik dan alamat logika adalah sama. Sedangkan pada waktu eksekusi menghasilkan alamat fisik dan alamat virtual yang berbeda.

Kumpulan alamat virtual yang dibuat oleh CPU disebut ruang alamat virtual. Kumpulan alamat fisik yang berkorespondensi dengan alamat virtual disebut ruang alamat fisik. Untuk mengubah alamat virtual ke alamat fisik diperlukan suatu perangkat keras yang bernama Memory Management Unit (MMU).

Register utamanya disebut register relokasi. Nilai pada register relokasi akan bertambah setiap alamat dibuat oleh proses pengguna dan pada waktu yang sama alamat ini dikirimkan ke memori. Ketika ada program yang menunjuk ke alamat memori, kemudian mengoperasikannya, dan menaruh lagi ke memori, akan dilokasikan oleh MMU karena program pengguna hanya berinteraksi dengan alamat logika. Pengubahan alamat virtual ke alamat fisik merupakan pusat dari manajemen memori.



Gambar 2 Relokasi dinamis menggunakan register relokasi

Pemanggilan Dinamis (Dynamic Loading)

Seluruh proses dan data berada di memori fisik ketika dieksekusi. Ukuran dari memori fisik terbatas. Untuk mendapatkan penggunaan ruang memori yang baik, kita melakukan pemanggilan secara dinamis. Dengan pemanggilan dinamis, sebuah rutin tidak akan dipanggil jika tidak diperlukan.

Semua rutin diletakkan dalam disk dengan format yang dapat dialokasikan ulang. Progam utama ditempatkan dalam memori dan dieksekusi. Jika sebuah rutin memanggil rutin lainnya, maka akan diperiksa terlebih dahulu apakah rutin tersebut ada di dalam memori atau tidak, jika tidak ada maka *linkage loader* akan dipanggil untuk menempatkan rutin-rutin yang diinginkan ke memori dan memperbaharui tabel alamat program untuk menyesuaikan perubahan. Kemudian kendali diberikan pada rutin yang baru dipanggil tersebut.

Keuntungan dari pemanggilan dinamis adalah rutin yang tidak digunakan tidak pernah dipanggil. Metode ini berguna untuk kode dalam jumlah banyak, ketika muncul kasus-kasus yang tidak lazim, seperti rutin yang salah. Dalam kode yang besar, walaupun ukuran kode besar, tapi yang dipanggil dapat jauh lebih kecil.

Pemanggilan Dinamis tidak memerlukan bantuan sistem operasi. Ini adalah tanggung jawab para pengguna untuk merancang program yang mengambil keuntungan dari metode ini. Sistem operasi dapat membantu pembuat program dengan menyediakan kumpulan data rutin untuk mengimplementasi pemanggilan dinamis.

Link Dinamis (Dynamic Linking)

Pada proses dengan banyak langkah, ditemukan juga penghubung-penghubung pustaka yang dinamis, yang menghubungkan semua rutin yang ada di pustaka. Beberapa sistem operasi hanya mendukung penghubungan yang statis, dimana seluruh rutin yang ada dihubungkan ke dalam suatu ruang alamat. Setiap program memiliki salinan dari seluruh pustaka. Konsep penghubungan dinamis, serupa dengan konsep pemanggilan dinamis. Pemanggilan lebih banyak ditunda selama waktu eksekusi, dari pada lama penundaan oleh penghubungan dinamis. Keistimewaan ini biasanya digunakan dalam sistem kumpulan pustaka, seperti pustaka bahasa subrutin. Tanpa fasilitas ini, semua program dalam sebuah sistem, harus mempunyai salinan dari pustaka bahasa mereka (atau setidaknya referensi rutin oleh program) termasuk dalam tampilan yang dapat dieksekusi.

Kebutuhan ini sangat boros baik untuk disk, maupun memori utama. Dengan pemanggilan dinamis, sebuah potongan dimasukkan ke dalam tampilan untuk setiap rujukan pustaka subrutin. Potongan ini adalah sebuah bagian kecil dari kode yang menunjukan bagaimana mengalokasikan pustaka rutin di memori dengan tepat, atau bagaimana menempatkan pustaka jika rutin belum ada.

Ketika potongan ini dieksekusi, dia akan memeriksa dan melihat apakah rutin yang dibutuhkan sudah ada di memori. Jika rutin yang dibutuhkan tidak ada di memori, program akan menempatkannya ke memori. Jika rutin yang dibutuhkan ada di memori, maka potongan akan mengganti dirinya dengan alamat dari rutin, dan mengeksekusi rutin. Berikutnya ketika segmentasi kode dicapai, rutin pada pustaka dieksekusi secara langsung, dengan begini tidak ada biaya untuk penghubungan dinamis. Dalam skema ini semua proses yang menggunakan sebuah kumpulan bahasa, mengeksekusi hanya satu dari salinan kode pustaka.

Fasilitas ini dapat diperluas menjadi pembaharuan pustaka. Sebuah kumpulan data dapat ditempatkan lagi dengan versi yang lebih baru dan semua program yang merujuk ke pustaka akan secara otomatis menggunakan versi yang baru. Tanpa pemanggilan dinamis, semua program akan membutuhkan pemanggilan kembali, untuk dapat mengakses pustaka yang baru. Jadi semua program tidak secara sengaja mengeksekusi yang baru, perubahan versi pustaka, informasi versi dapat dimasukkan ke dalam memori, dan setiap program menggunakan informasi versi untuk memutuskan versi mana yang akan digunakan dari salinan pustaka. Sedikit perubahan akan tetap meggunakan nomor versi yang sama, sedangkan perubahan besar akan menambah satu versi sebelumnya. Karenanya program yang dikompilasi dengan versi yang baru akan dipengaruhi dengan perubahan yang terdapat di dalamnya. Program lain yang berhubungan sebelum pustaka baru diinstal, akan terus menggunakan pustaka lama. Sistem ini juga dikenal sebagai berbagi pustaka. Jadi seluruh pustaka yang ada dapat digunakan bersama-sama. Sistem seperti ini membutuhkan bantuan sistem operasi.

Overlays

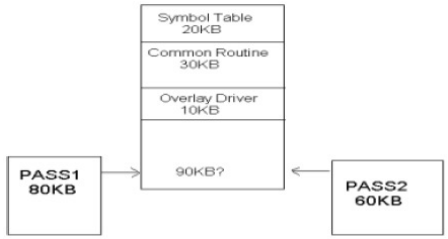
Overlays merupakan suatu metode untuk memungkinkan suatu proses yang membutuhkan memori yang cukup besar menjadi lebih sederhana. Penggunaan overlays ini dapat menghemat memori yang digunakan dalam pengeksekusian instruksi-instruksi. Hal ini sangat berguna terlebih jika suatu program yang ingin dieksekusi mempunyai ukuran yang lebih besar daripada alokasi memori yang tersedia.

Cara kerjanya yaitu pertama-tama membuat beberapa overlays yang didasarkan pada instruksi-instruksi yang dibutuhkan pada satu waktu tertentu. Setelah itu, membuat overlays drivernya yang digunakan sebagai jembatan atau perantara antara overlays yang dibuat. Proses selanjutnya ialah me-load instruksi yang dibutuhkan pada satu waktu ke dalam memori absolut dan menunda instruksi lain yang belum di butuhkan pada saat itu. Setelah selesai dieksekusi maka instruksi yang tertunda akan diload menggantikan instruksi yang sudah tidak dibutuhkan lagi.

Proses-proses tersebut diatur oleh overlay driver yang dibuat oleh pengguna. Untuk membuat suatu overlays dibutuhkan relokasi dan linking algoritma yang baik oleh pengguna. Untuk lebih jelasnya perhatikan gambar dibawah ini.

Sebagai contoh, ada beberapa instruksi seperti pada gambar diatas. Untuk menempatkan semuanya sekaligus, kita akan membutuhkan 140K memori. Jika hanya 90K yang tersedia, kita tidak dapat menjalankan proses. Perhatikan bahwa PASS1 dan PASS2 tidak harus berada di memori pada saat yang sama. Kita mendefinisikan dua buah overlays. Overlays A untuk PASS1, tabel simbol dan rutin, overlays B untuk PASS2, tabel simbol dan rutin. Lalu kita buat sebuah overlay driver (10 Kbytes)

sebagai jembatan antara kedua overlays tersebut. Pertama-tama kita mulai dengan me-load overlays A ke memori. Setelah dieksekusi, kemudian pindah ke overlay driver, me-load overlays B ke dalam memori, menimpa overlays A, dan mengirim kendali ke PASS2. Overlays A hanya butuh 80 Kb, dan overlays B membutuhkan 60 Kb memori. Nah, sekarang kita dapat menjalankan program dengan memori 140 Kb (karena kita menjalankannya secara bergantian). Seperti dalam pemanggilan dinamis, overlays tidak membutuhkan bantuan dari sistem operasi. Implementasi dapat dilakukan sepenuhnya oleh pengguna, oleh karenanya programmer harus merancang overlays tersebut dengan algoritma yang tepat.



Gambar 3 Overlay untuk 2-PASS assembler

Swapping

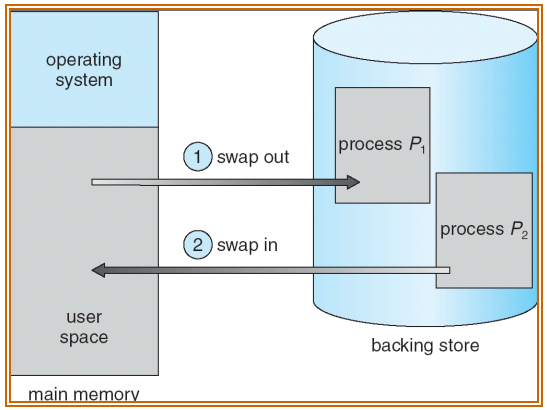
Sebuah proses agar bisa dieksekusi bukan hanya membutuhkan sumber daya dari CPU, tetapi juga harus terletak dalam memori. Dalam tahapannya, suatu proses bisa saja ditukar sementara keluar memori ke sebuah penyimpanan sementara dan kemudian dibawa lagi ke memori untuk melanjutkan pengeksekusian. Hal ini dalam sistem operasi disebut swapping. Sebagai contoh, asumsikan sebuah multiprogramming environment dengan algoritma penjadwalan CPU round-robin. Ketika waktu kuantum habis, pengatur memori akan menukar proses yang telah selesai dan memasukkan proses yang lain ke dalam memori yang sudah bebas. Sementara di saat yang bersamaan, penjadwal CPU akan mengalokasikan waktu untuk proses lain di dalam memori. Ketika waktu kuantum setiap proses sudah habis, proses tersebut akan ditukar dengan proses lain. Untuk kondisi yang ideal, manajer memori dapat melakukan penukaran proses dengan cepat sehingga proses akan selalu berada dalam memori dan siap dieksekusi saat penjadwal CPU hendak menjadwal CPU. Hal ini juga berkaitan dengan CPU utilization.

Swaping dapat juga kita lihat dalam algoritma berbasis prioritas. Jika proses dengan prioritas lebih tinggi tiba dan meminta layanan, manajer memori dapat menukar keluar memori proses-proses yang prioritasnya rendah sehingga proses-proses yang prioritasnya lebih tinggi tersebut dapat dieksekusi. Setelah proses-proses yang memiliki prioritas lebih tinggi tersebut selesai dieksekusi, proses-proses dengan prioritas rendah dapat ditukar kembali ke dalam memori dan dilanjutkan eksekusinya. Cara ini disebut juga dengan metode ***roll in***, ***roll out***.

Ketika proses yang sebelumnya ditukar, akan dikembalikan ke ruang memori. Ada 2 kemungkinan yang terjadi. Pertama, apabila pemberian alamat dilakukan pada waktu pembuatan atau waktu pengambilan, maka proses tersebut pasti akan menempati ruang memori yang sama. Akan tetapi, apabila pemberian alamat diberikan pada waktu eksekusi, ada kemungkinan proses akan dikembalikan ke ruang memori yang berbeda dengan sebelumnya. Dua kemungkinan ini berkaitan dengan penjelasan pada bab sebelumnya yaitu Manajemen Memori.

Penukaran membutuhkan sebuah penyimpanan sementara. Penyimpanan sementara pada umumnya adalah sebuah *fast disk*, dan harus cukup untuk menampung salinan dari seluruh gambaran memori untuk semua pengguna, dan harus mendukung akses langsung terhadap gambaran memori tersebut. Sistem mengatur *ready queue* yang berisikan semua proses yang gambaran memorinya berada di memori dan siap untuk dijalankan. Saat sebuah penjadwal CPU ingin menjalankan sebuah proses, ia akan memeriksa apakah proses yang mengantri di *ready queue* tersebut sudah berada di dalam memori tersebut atau belum. Apabila belum, penjadwal CPU akan melakukan penukaran keluar terhadap proses-proses yang berada di dalam memori sehingga tersedia tempat untuk memasukkan

proses yang hendak dieksekusi tersebut. Setelah itu *register* dikembalikan seperti semula dan proses yang diinginkan akan dieksekusi.



Gambar 4 Skema view dari Swapping

Waktu pergantian isi dalam sebuah sistem yang melakukan penukaran pada umumnya cukup tinggi. Untuk mendapatkan gambaran mengenai waktu pergantian isi, akan diilustrasikan sebuah contoh. Misalkan ada sebuah proses sebesar 1 MB, dan media yang digunakan sebagai penyimpanan sementara adalah sebuah hard disk dengan kecepatan transfer 5 MBps. Waktu yang dibutuhkan untuk mentransfer proses 1 MB tersebut dari atau ke dalam memori adalah:

1000 KB/5000 KBps = 1/5 detik = 200 milidetik

Apabila diasumsikan head seek tidak dibutuhkan dan rata-rata waktu latensi adalah 8 milidetik, satu proses penukaran memakan waktu 208 milidetik. Karena kita harus melakukan proses penukaran sebanyak 2 kali, (memasukkan dan mengeluarkan dari memori), maka keseluruhan waktu yang dibutuhkan adalah 416 milidetik.

Untuk penggunaan CPU yang efisien, kita menginginkan waktu eksekusi kita relatif panjang dibandingkan dengan waktu penukaran. Oleh karena itu, misalnya dalam penjadwalan CPU yang menggunakan metoda round robin, waktu kuantum yang kita tetapkan harus lebih besar dari 416 milidetik. Jika tidak, waktu lebih banyak terbuang pada proses penukaran saja sehingga penggunaan prosesor tidak efisien lagi.

Bagian utama dari waktu penukaran adalah waktu transfer. Besar waktu transfer berhubungan langsung dengan jumlah memori yang di-tukar. Jika kita mempunyai sebuah komputer dengan memori utama 128 MB dan sistem operasi memakan tempat 5 MB, besar proses pengguna maksimal adalah 123 MB. Bagaimana pun juga, proses pengguna pada kenyataannya dapat berukuran jauh lebih kecil dari angka tersebut. Bahkan terkadang hanya berukuran 1 MB. Proses sebesar 1 MB dapat ditukar hanya dalam waktu 208 milidetik, jauh lebih cepat dibandingkan menukar proses sebesar 123 MB yang akan menghabiskan waktu 24.6 detik. Oleh karena itu, sangatlah berguna apabila kita mengetahui dengan baik berapa besar memori yang dipakai oleh proses pengguna, bukan sekedar dengan perkiraan saja. Setelah itu, kita dapat mengurangi besar waktu penukaran dengan cara hanya menukar proses-proses yang benar-benar membutuhkannya.

Agar metoda ini bisa dijalankan dengan efektif, pengguna harus menjaga agar sistem selalu memiliki informasi mengenai perubahan kebutuhan memori. Oleh karena itu, proses yang membutuhkan memori dinamis harus melakukan pemanggilan sistem (permintaan memori dan pelepasan memori) untuk memberikan informasi kepada sistem operasi akan perubahan kebutuhan memori.

Penukaran dipengaruhi oleh banyak faktor. Jika kita hendak menukar suatu proses, kita harus yakin bahwa proses tersebut siap. Hal yang perlu diperhatikan adalah kemungkinan proses tersebut sedang

menunggu I/O. Apabila I/O secara asinkron mengakses memori pengguna untuk I/O buffer, maka proses tersebut tidak dapat ditukar. Apabila sebuah operasi I/O berada dalam antrian karena peralatan I/O-nya sedang sibuk. Kemudian kita hendak mengeluarkan proses P1 dan memasukkan proses P2. Operasi I/O mungkin akan berusaha untuk memakai memori yang sekarang seharusnya akan ditempati oleh P2. Cara untuk mengatasi masalah ini adalah:

1. Menghindari penukaran proses yang sedang menunggu I/O.
2. Melakukan eksekusi operasi I/O hanya di buffer sistem operasi.

Hal tersebut akan menjaga agar transfer antara buffer sistem operasi dan proses memori hanya terjadi saat si proses ditukar kedalam.

Pada saat ini, proses penukaran secara dasar hanya digunakan di sedikit sistem. Hal ini dikarenakan penukaran menghabiskan terlalu banyak waktu tukar dan memberikan waktu eksekusi yang terlalu kecil sebagai solusi dari manajemen memori. Akan tetapi, banyak sistem yang menggunakan versi modifikasi dari metode penukaran ini.

Salah satu sistem operasi yang menggunakan versi modifikasi dari metoda penukaran ini adalah UNIX. Penukaran berada dalam keadaan non-aktif, sampai apabila ada banyak proses yang berjalan yang menggunakan memori yang besar. Penukaran akan berhenti lagi apabila jumlah proses yang berjalan sudah berkurang.

Pada awal pengembangan komputer pribadi, tidak banyak perangkat keras (atau sistem operasi yang memanfaatkan perangkat keras) yang dapat mengimplementasikan memori manajemen yang baik, melainkan digunakan untuk menjalankan banyak proses berukuran besar dengan menggunakan versi modifikasi dari metoda penukaran. Salah satu contoh yang baik adalah Microsoft Windows 3.1, yang mendukung eksekusi proses berkesinambungan. Apabila suatu proses baru hendak dijalankan dan tidak terdapat cukup memori, proses yang lama perlu dimasukkan ke dalam disk. Sistem operasi ini, bagaimana pun juga, tidak mendukung penukaran secara keseluruhan karena yang lebih berperan menentukan kapan proses penukaran akan dilakukan adalah pengguna dan bukan penjadwal CPU. Proses-proses yang sudah dikeluarkan akan tetap berada di luar memori sampai pengguna memilih proses yang hendak dijalankan. Sistem-sistem operasi Microsoft selanjutnya, seperti misalnya Windows NT, memanfaatkan fitur Unit Manajemen Memori.

Proteksi Memori

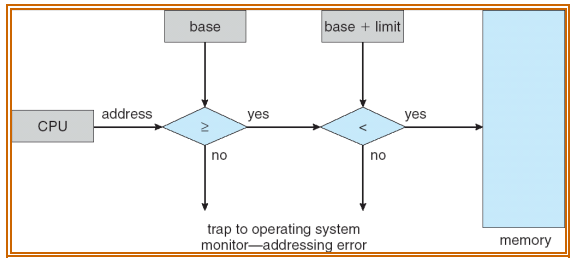
Proteksi memori adalah sebuah sistem yang mencegah sebuah proses dari pengambilan memori proses lain yang sedang berjalan pada komputer yang sama dan pada saat yang sama pula. Proteksi memori selalu mempekerjakan hardware (Memori Manajemen Unit/MMU) dan sistem software untuk mengalokasikan memori yang berbeda untuk proses yang berbeda dan untuk mengatasi *exception* yang muncul ketika sebuah proses mencoba untuk mengakses memori di luar batas.

Proteksi memori dapat menggunakan Relocation Register dengan Limit Register. Relocation Register berisi nilai terkecil alamat fisik. Limit Register berisi rentang nilai alamat logika. Dengan Relokasi dan Limit Register, tiap alamat logika harus lebih kecil dari Limit Register. MMU memetakan alamat logika secara dinamis dengan menambahkan nilai di Relocation Register. Alamat pemetaan ini kemudian dikirimkan ke memori.

Efektifitas dari proteksi memori berbeda antara sistem operasi yang satu dengan yang lainnya. Ada beberapa cara yang berbeda untuk mencapai proteksi memori. Segmentasi dan pemberian halaman adalah dua metoda yang paling umum digunakan. Segmentasi adalah skema manajemen memori dengan cara membagi memori menjadi segmen-segmen. Dengan demikian, sebuah program dibagi menjadi segmen-segmen. Segmen adalah sebuah unit logis, yaitu unit yang terdiri dari beberapa bagian yang berjenis yang sama.

Segmen dapat terbagi jika terdapat elemen di tabel segmen yang berasal dari dua proses yang berbeda yang menunjuk pada alamat fisik yang sama. Saling berbagi ini muncul di level segmen dan pada saat ini terjadi semua informasi dapat turut terbagi. Proteksi dapat terjadi karena ada bit proteksi yang berhubungan dengan setiap elemen dari segmen tabel. Bit proteksi ini berguna untuk mencegah akses ilegal ke memori. Caranya menempatkan sebuah array di dalam segmen itu sehingga perangkat keras manajemen memori secara otomatis akan memeriksa indeks arraynya legal atau tidak.

Pemberian halaman merupakan metode yang paling sering digunakan untuk proteksi memori. Pemberian halaman adalah suatu metoda yang memungkinkan suatu alamat fisik memori yang tersedia dapat tidak berurutan. Proteksi memori di lingkungan halaman bisa dilakukan dengan cara memproteksi bit-bit yang berhubungan dengan setiap frame. Biasanya bit-bit ini disimpan di dalam sebuah tabel halaman. Satu bit bisa didefinisikan sebagai baca-tulis atau hanya baca saja. Setiap referensi ke memori menggunakan tabel halaman untuk menemukan nomor frame yang benar. Pada saat alamat fisik sedang dihitung, bit proteksi bisa memeriksa bahwa kita tidak bisa menulis ke mode tulis saja.



Gambar 5 Dukungan perangkat keras untuk relokasi dan pembatasan register-register

Alokasi Memori Berkesinambungan

Alokasi memori berkesinambungan berarti alamat memori diberikan kepada proses secara berurutan dari kecil ke besar. Keuntungan menggunakan alokasi memori berkesinambungan dibandingkan menggunakan alokasi memori tidak berkesinambungan adalah:

a. Sederhana
b. Cepat
c. Mendukung proteksi memori

Sedangkan kerugian dari menggunakan alokasi memori berkesinambungan adalah apabila tidak semua proses dialokasikan di waktu yang sama, akan menjadi sangat tidak efektif sehingga mempercepat habisnya memori.

Alokasi memori berkesinambungan dapat dilakukan baik menggunakan sistem partisi banyak, maupun menggunakan sistem partisi tunggal. Sistem partisi tunggal berarti alamat memori yang akan dialokasikan untuk proses adalah alamat memori pertama setelah pengalokasian sebelumnya. Sedangkan sistem partisi banyak berarti sistem operasi menyimpan informasi tentang semua bagian memori yang tersedia untuk dapat diisi oleh proses-proses (disebut lubang). Sistem partisi banyak kemudian dibagi lagi menjadi sistem partisi banyak tetap, dan sistem partisi banyak dinamis. Hal yang membedakan keduanya adalah untuk sistem partisi banyak tetap, memori dipartisi menjadi blok-blok yang ukurannya tetap yang ditentukan dari awal. Sedangkan sistem partisi banyak dinamis artinya memori dipartisi menjadi bagian-bagian dengan jumlah dan besar yang tidak tentu.

Sistem partisi banyak

Sistem operasi menyimpan sebuah tabel yang menunjukkan bagian mana dari memori yang memungkinkan untuk menyimpan proses, dan bagian mana yang sudah diisi. Pada intinya, seluruh memori dapat diisi oleh proses pengguna. Saat sebuah proses datang dan membutuhkan memori, CPU akan mencari lubang yang cukup besar untuk menampung proses tersebut. Setelah menemukannya, CPU akan mengalokasikan memori sebanyak yang dibutuhkan oleh proses tersebut, dan mempersiapkan sisanya untuk menampung proses-proses yang akan datang kemudian (seandainya ada).

Saat proses memasuki sistem, proses akan dimasukkan ke dalam antrian masukan. Sistem operasi akan menyimpan besar memori yang dibutuhkan oleh setiap proses dan jumlah memori kosong yang tersedia, untuk menentukan proses mana yang dapat diberikan alokasi memori. Setelah sebuah proses mendapat alokasi memori, proses tersebut akan dimasukkan ke dalam memori. Setelah

proses tersebut dimatikan, proses tersebut akan melepas memori tempat dia berada, yang mana dapat diisi kembali oleh proses lain dari antrian masukan.

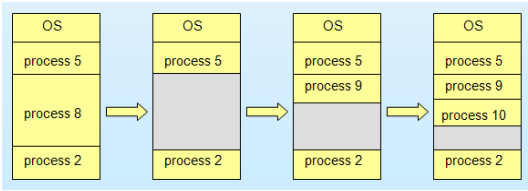
Sistem operasi setiap saat selalu memiliki catatan jumlah memori yang tersedia dan antrian masukan. Sistem operasi dapat mengatur antrian masukan berdasarkan algoritma penjadwalan yang digunakan. Memori dialokasikan untuk proses sampai akhirnya kebutuhan memori dari proses selanjutnya tidak dapat dipenuhi (tidak ada lubang yang cukup besar untuk menampung proses tersebut). Sistem operasi kemudian dapat menunggu sampai ada blok memori cukup besar yang kosong, atau dapat mencari proses lain di antrian masukan yang kebutuhan memorinya memenuhi jumlah memori yang tersedia.

Pada umumnya, kumpulan lubang-lubang dalam berbagai ukuran tersebar di seluruh memori sepanjang waktu. Apabila ada proses yang datang, sistem operasi akan mencari lubang yang cukup besar untuk menampung memori tersebut. Apabila lubang yang tersedia terlalu besar, akan dipecah menjadi 2. Satu bagian akan dialokasikan untuk menerima proses tersebut, sementara bagian lainnya tidak digunakan dan siap menampung proses lain. Setelah proses selesai, proses tersebut akan melepas memori dan mengembalikannya sebagai lubang-lubang. Apabila ada dua lubang yang kecil yang berdekatan, keduanya akan bergabung untuk membentuk lubang yang lebih besar. Pada saat ini, sistem harus memeriksa apakah ada proses yang menunggu yang dapat dimasukkan ke dalam ruang memori yang baru terbentuk tersebut.

Hal ini disebut Permasalahan alokasi penyimpanan dinamis, yakni bagaimana memenuhi permintaan sebesar n dari kumpulan lubang-lubang yang tersedia. Ada berbagai solusi untuk mengatasi hal ini, yaitu:

1. ***First fit***: Mengalokasikan lubang pertama ditemukan yang besarnya mencukupi. Pencarian dimulai dari awal.
2. ***Best fit***: Mengalokasikan lubang dengan besar minimum yang mencukupi permintaan.
3. ***Next fit***: Mengalokasikan lubang pertama ditemukan yang besarnya mencukupi. Pencarian dimulai dari akhir pencarian sebelumnya.
4. ***Worst fit***: Mengalokasikan lubang terbesar yang ada.

Memilih yang terbaik diantara keempat metode di atas adalah sepenuhnya tergantung kepada pengguna, karena setiap metode memiliki kelebihan dan kekurangan masing-masing. Menggunakan *best fit* dan *worst fit* berarti kita harus selalu memulai pencarian lubang dari awal, kecuali apabila lubang sudah disusun berdasarkan ukuran. Metode *worst fit* akan menghasilkan sisa lubang yang terbesar, sementara metoda *best fit* akan menghasilkan sisa lubang yang terkecil.



Gambar 6 Alokasi memori partisi banyak

Fragmentasi

Fragmentasi adalah munculnya lubang-lubang yang tidak cukup besar untuk menampung permintaan dari proses. Fragmentasi dapat berupa fragmentasi internal maupun fragmentasi eksternal. *Fragmentasi eksternal* muncul apabila jumlah keseluruhan memori kosong yang tersedia memang mencukupi untuk menampung permintaan tempat dari proses, tetapi letaknya tidak berkesinambungan atau terpecah menjadi beberapa bagian kecil sehingga proses tidak dapat masuk.

Umumnya, ini terjadi ketika kita menggunakan sistem partisi banyak dinamis. Pada sistem partisi banyak dinamis, seperti yang diungkapkan sebelumnya, sistem terbagi menjadi blok-blok yang besarnya tidak tetap. Maksud tidak tetap di sini adalah blok tersebut bisa bertambah besar atau bertambah kecil. Misalnya, sebuah proses meminta ruang memori sebesar 17 KB, sedangkan memori dipartisi menjadi blok-blok yang besarnya masing-masing 5 KB. Maka, yang akan diberikan pada proses adalah 3 blok ditambah 2 KB dari sebuah blok. Sisa blok yang besarnya 3 KB akan disiapkan

untuk menampung proses lain atau jika ia bertetangga dengan ruang memori yang kosong, ia akan bergabung dengannya. Akibatnya dengan sistem partisi banyak dinamis, bisa tercipta lubang-lubang di memori, yaitu ruang memori yang kosong. Keadaan saat lubang-lubang ini tersebar yang masing-masing lubang tersebut tidak ada yang bisa memenuhi kebutuhan proses padahal jumlah dari besarnya lubang tersebut cukup untuk memenuhi kebutuhan proses disebut sebagai *fragmentasi eksternal*.

Fragmentasi internal muncul apabila jumlah memori yang diberikan oleh penjadwal CPU untuk ditempati proses lebih besar daripada yang diminta proses karena adanya selisih antara permintaan proses dengan alokasi lubang yang sudah ditetapkan. Hal ini umumnya terjadi ketika kita menggunakan sistem partisi banyak tetap. Kembali ke contoh sebelumnya, di mana ada proses dengan permintaan memori sebesar 17 KB dan memori dipartisi menjadi blok yang masing-masing besarnya 5 KB. Pada sistem partisi banyak tetap, memori yang dialokasikan untuk proses adalah 4 blok, atau sebesar 20 KB. Padahal, yang terpakai hanya 17 KB. Sisa 3 KB tetap diberikan pada proses tersebut, walaupun tidak dipakai oleh proses tersebut. Hal ini berarti pula proses lain tidak dapat memakainya. Perbedaan memori yang dialokasikan dengan yang diminta inilah yang disebut *fragmentasi internal*.

Algoritma alokasi penyimpanan dinamis mana pun yang digunakan, tetap tidak bisa menutup kemungkinan terjadinya fragmentasi. Bahkan hal ini bisa menjadi fatal. Salah satu kondisi terburuk adalah apabila kita memiliki memori terbuang setiap dua proses. Apabila semua memori terbuang itu digabungkan, bukan tidak mungkin akan cukup untuk menampung sebuah proses. Sebuah contoh statistik menunjukkan bahwa saat menggunakan metoda first fit, bahkan setelah dioptimisasi, dari N blok teralokasi, sebanyak 0.5N blok lain akan terbuang karena fragmentasi. Jumlah sebanyak itu berarti kurang lebih setengah dari memori tidak dapat digunakan. Hal ini disebut dengan aturan 50%. Fragmentasi ekstern dapat diatasi dengan beberapa cara, diantaranya adalah:

1. Pemadatan, yaitu mengatur kembali isi memori agar memori yang kosong diletakkan bersama di suatu bagian yang besar sehingga proses dapat masuk ke ruang memori kosong tersebut. Pemadatan hanya dapat dilakukan bila pengalamatan program dilakukan pada saat eksekusi. Kebutuhan akan pemadatan akan hilang bila pengalamatan dapat dilakukan secara berurutan, walaupun sebenarnya proses tidak ditempatkan pada lokasi memori yang berurutan. Nantinya, konsep ini diterapkan dengan Paging.
2. Penghalamanan.
3. Segmentasi.

Fragmentasi internal tidak dapat dihindarkan apabila kita menggunakan sistem partisi banyak berukuran tetap, mengingat besar *hole* yang disediakan selalu tetap, kecuali jika kita menggunakan sistem partisi banyak dinamis, yang memungkinkan suatu proses untuk diberikan ruang memori sebesar yang dia minta.

Penghalaman Memori

Penghalaman merupakan metode yang memungkinkan suatu alamat fisik memori yang tersedia dapat tidak berurutan. Pemberian halaman bisa menjadi solusi untuk pemecahan masalah luar. Untuk bisa mengimplementasikan solusi ini adalah melalui pengunaan dari skema pemberian halaman. Dengan pemberian halaman bisa mencegah masalah penting dari pemuatan besar ukuran memori yang bervariasi kedalam penyimpanan cadangan. Ketika beberapa pecahan kode dari data yang tersisa di memori utama perlu untuk ditukar keluar, harus ditemukan ruang untuk penyimpanan cadangan. Masalah pemecahan kode didiskusikan dengan kaitan bahwa pengaksesannya lebih lambat. Biasanya bagian yang menunjang untuk pemberian halaman telah ditangani oleh perangkat keras. Bagaimana pun, desain yang ada baru-baru ini telah diimplementasikan dengan menggabungkan perangkat keras dan sistem operasi, terutama pada microprocessor 64 bit .

Keuntungan dan kerugian pemberian halaman adalah:
Jika kita membuat ukuran dari masing-masing halaman menjadi lebih besar.

- Keuntungan. Akses memori akan relatif lebih cepat.
- Kerugian. Kemungkinan terjadinya fragmentasi intern sangat besar.

Jika kita membuat ukuran dari masing-masing halaman menjadi lebih kecil.

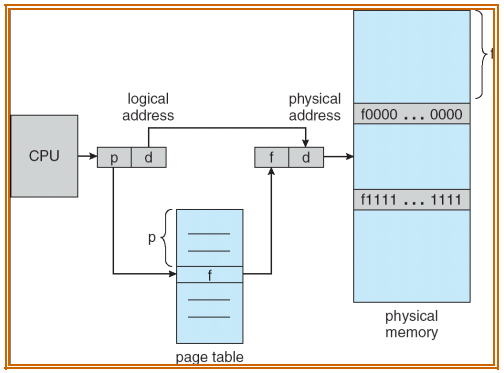
- Keuntungan. Kemungkinan terjadinya internal Framentasi akan menjadi lebih kecil.
- Kerugian. Akses memori akan relatif lebih lambat.

Keuntungan lainnya dari paging adalah, konsep memori virtual bisa diterapkan dengan menuliskan halaman ke disk, dan pembacaan halaman dari disk ketika dibutuhkan. Hal ini dikarenakan jarangnya pengunaan kode-kode dan data suatu program secara keseluruhan pada suatu waktu.

Kerugian lainnya dari paging adalah, paging tidak bisa diterapkan untuk beberapa prosesor tua atau kecil (dalam keluarga Intel x86, sebagai contoh, hanya 80386 dan di atasnya yang punya MMU, yang bisa diterapkan paging). Hal ini dikarenakan paging membutuhkan MMU (Memory Management Unit).

Metode Dasar

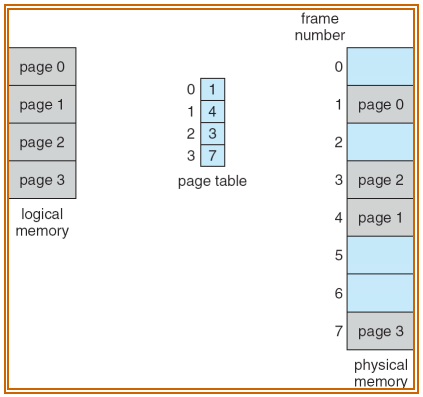
Jadi metode dasar yang digunakan adalah dengan memecah memori fisik menjadi blok-blok berukuran tetap yang akan disebut sebagai frame. Selanjutnya memori logis juga dipecah menjadi blok-blok dengan ukuran yang sama disebut sebagai halaman. Selanjutnya pembuatan suatu tabel halaman yang akan menerjemahkan memori logis kita ke dalam memori fisik. Jika suatu proses ingin dieksekusi maka memori logis akan melihat dimanakah dia akan ditempatkan di memori fisik dengan melihat kedalam tabel halamannya.



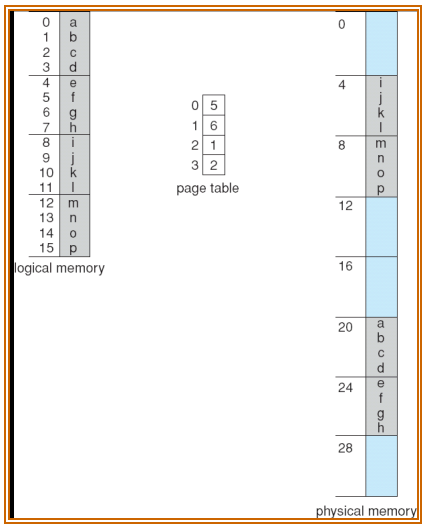
Gambar 7 Penghalaman dengan perangkat keras

Setiap alamat yang dihasilkan oleh CPU dibagi-bagi menjadi dua bagian yaitu sebuah nomor halaman (p) dan sebuah offset halaman (d). Nomor halaman ini akan digunakan sebagai indeks untuk tabel halaman. Tabel halaman mengandung basis alamat dari tiap-tiap halaman di memori fisik. Basis ini dikombinasikan dengan offset halaman untuk menentukan alamat memori fisik yang dikirim ke unit memori.

Memori fisik dipecah menjadi beberapa blok berukuran yang tetap yang disebut frame (frame), sedangkan memori logis juga dipecah dengan ukuran yang sama yang disebut halaman. Suatu alamat memori yang di-*generate* oleh CPU terdiri dari 2 bagian yaitu halaman dan offset. Halaman berfungsi sebagai indeks dari suatu tabel halaman. Isi dari indeks yang ditunjuk pada tabel halaman (*frame*) digabungkan dengan offset maka akan membentuk suatu alamat asli dari suatu data pada memori fisik. Offset sendiri berfungsi sebagai penunjuk dari suatu blok data yang berada dalam suatu *frame*.



Gambar 8 Model penghalaman memori logik dan fisik



Gambar 9 Contoh penghalaman untuk memori 32-bit dengan 4 byte per halaman

Dukungan Perangkat Keras

Setiap sistem operasi mempunyai caranya tersendiri untuk menyimpan tabel halaman. Biasanya sistem operasi mengalokasikan sebuah tabel halaman untuk setiap proses. sebuah penunjuk ke tabel halaman disimpan dengan nilai register yang lain didalam blok pengontrol proses. Akan tetapi penggunaannya menjadi tidak praktis karena ternyata tabel halaman disimpan pada memori utama yang tentu saja kecepatannya jauh lebih lambat dari dari register.

Translation Lookaside Buffers (TLBs) dibuat untuk mengatasi masalah tersebut. TLBs adalah suatu asosiatif memori berkecepatan tinggi yang berfungsi hampir sama seperti cache memori tapi terjadi pada tabel halaman, TLBs menyimpan sebagian alamat-alamat data dari suatu proses yang berada pada tabel halaman yang sedang digunakan atau sering digunakan. TLBs biasanya terletak pada Memori Management Unit (MMU).

Salah satu feature dari TLBs adalah mampu membuat proteksi suatu alamat memori. Fitur ini dinamakan address-space identifiers (ASIDs). ASIDs ini membuat suatu alamat memori hanya ada 1 proses yang bisa mengaksesnya. Contohnya adalah JVM. Pada saat program Java berjalan, Java membuat alokasi alamat memori untuk JVM dan yang bisa mengakses alamat tersebut hanya program Java yang sedang berjalan itu saja.
Suatu keadaan dimana pencarian alamat memori berhasil ditemukan pada TLBs disebut TLB hit. Sedangkan sebaliknya jika terjadi pada tabel halaman (dalam hal ini TLB gagal) disebut TLB miss. Effective Address Time adalah waktu yang dibutuhkan untuk mengambil data dalam memori fisik dengan persentase TLB hit (hit ratio) dan TLB miss (miss ratio).

Persentasi dari beberapa kali TLB hit adalah disebut hit ratio. hit ratio 80% berarti menemukan nomor halaman yang ingin kita cari didalam TLB sebesar 80%. Jika waktu akses ke TLB memakan waktu 20 nanodetik dan akses ke memori memakan waktu sebesar 100 nanodetik maka total waktu kita memetakan memori adalah 120 nanodetik jika TLB hit. dan jika TLB miss maka total waktunya adalah 220 nanodetik. Jadi untuk mendapatkan waktu akses memori yang efektif maka kita harus membagi-bagi tiap kasus berdasarkan kemungkinannya:

```
Effective address time = hit ratio * (time search TLB + time access memori)
+ miss ratio * (time search TLB + time access tabel halaman + time access
memori)
```

Suatu mesin yang efektif apabila memiliki effective address time yang kecil. Banyak cara yang telah dilakukan untuk optimal, salah satunya dengan mereduksi TLBs miss. Jadi sistem operasi memiliki intuisi untuk memprediksi halaman selanjutnya yang akan digunakan kemudian me-loadnya ke dalam TLB.

Proteksi Memori

Proteksi memori di lingkungan halaman dikerjakan oleh bit-bit proteksi yang berhubungan dengan tiap frame. Biasanya bit-bit ini disimpan didalam sebuah tabel halaman. Satu bit bisa didefinisikan sebagai baca-tulis atau hanya baca saja suatu halaman. Setiap referensi ke memori menggunakan tabel halaman untuk menemukan nomor frame yang benar. Pada saat alamat fisik sedang dihitung, bit proteksi bisa dicek untuk memastikan tidak ada kegiatan tulis-menulis ke dalam 'read-only halaman'. Hal ini diperlukan untuk menghindari tejadinya '*memori-protection violation*', suatu keadaan dimana terjadi suatu percobaan menulis di 'halaman read-only'.

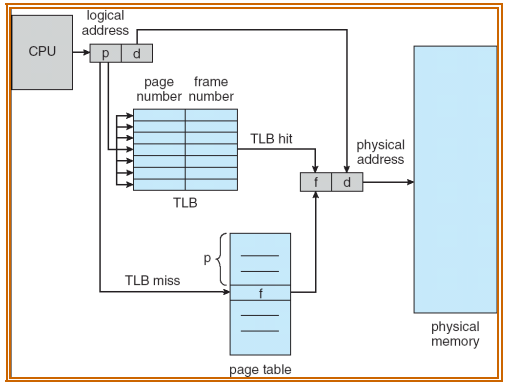
Ketika bit proteksi bernilai 'valid', berarti halaman yang dicari ada di dalam ruang alamat logika. Yang artinya halaman tersebut dapat diakses. Jika bit proteksi bernilai 'invalid', artinya halaman yang dimaksud tidak berada dalam ruang alamat logika. Sehingga halaman tersebut tidak dapat diakses.

Tabel Halaman

Sebagian besar komputer modern memiliki perangkat keras istimewa yaitu unit manajemen memori (MMU). Unit tersebut berada diantara CPU dan unit memori. Jika CPU ingin mengakses memori (misalnya untuk memanggil suatu instruksi atau memanggil dan menyimpan suatu data), maka CPU mengirimkan alamat memori yang bersangkutan ke MMU, yang akan menerjemahkannya ke alamat lain sebelum melanjutkannya ke unit memori. Alamat yang dihasilkan oleh CPU, setelah adanya pemberian indeks atau aritmatik ragam pengalamatan lainnya disebut alamat logis (virtual address). Sedangkan alamat yang didapatkan fisik membuat sistem operasi lebih mudah pekerjaannya saat mengalokasikan memori. Lebih penting lagi, MMU juga mengizinkan halaman yang tidak sering digunakan bisa disimpan di disk. Cara kerjanya adalah sebagai berikut:

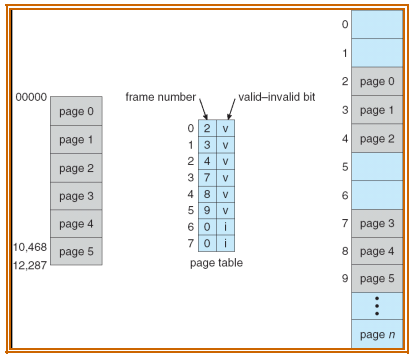
Tabel yang digunakan oleh MMU mempunyai bit sahih untuk setiap halaman di bagian alamat logis. Jika bit tersebut di set, maka penterjemahan oleh alamat logis di halaman itu berjalan normal. Akan tetapi jika dihapus, adanya usaha dari CPU untuk mengakses suatu alamat di halaman tersebut menghasilkan suatu interupsi yang disebut page *fault trap*. Sistem operasi telah mempunyai interrupt handler untuk kesalahan halaman, juga bisa digunakan untuk mengatasi interupsi jenis yang lain. Handler inilah yang akan bekerja untuk mendapatkan halaman yang diminta ke memori. Untuk lebih jelasnya, saat kesalahan halaman dihasilkan untuk halaman p1, interrupt handler melakukan hal-hal berikut ini:

- Mencari dimana isi dari halaman p1 disimpan di disk. Sistem operasi menyimpan informasi ini di dalam tabel. Ada kemungkinan bahwa halaman tersebut tidak ada dimana-mana, misalnya pada kasus saat referensi memori adalah bug. Pada kasus tersebut, sistem operasi mengambil beberapa langkah kerja seperti mematikan prosesnya. Dan jika diasumsikan halamannya berada dalam disk:
- Mencari halaman lain yaitu p2 yang dipetakan ke frame lain f dari alamat fisik yang tidak banyak dipergunakan.
- Menyalin isi dari frame f keluar dari disk.
- Menghapus bit sahih dari halaman p2 sehingga sebagian referensi dari halaman p2 akan menyebabkan kesalahan halaman.
- Menyalin data halaman p1 dari disk ke frame f.
- Update tabel MMU sehingga halaman p1 dipetakan ke frame f.
- Kembali dari interupsi dan mengizinkan CPU mengulang instruksi yang menyebabkan interupsi tersebut.



Gambar 10 Penghalaman perangkat keras dengan TLB

Pada dasarnya MMU terdiri dari tabel halaman yang merupakan sebuah rangkaian array dari masukan-masukan (entries) yang mempunyai indeks berupa nomor halaman (p). Setiap masukan terdiri dari flags (contohnya bit sahih dan nomor frame). Alamat fisik dibentuk dengan menggabungkan nomor frame dengan offset, yaitu bit paling rendah dari alamat logis.

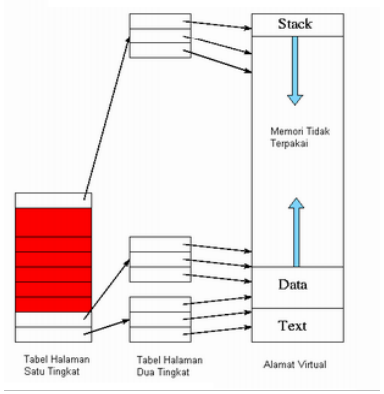


Gambar 11 Skema tabel halaman dua tingkat

Pemberian Halaman Secara Meningkat

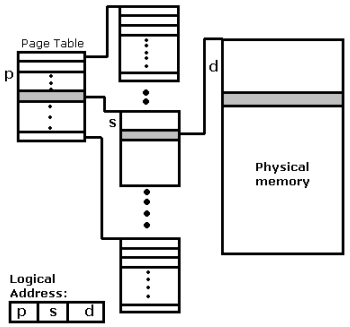
Hierarchical paging atau pemberian halaman bertingkat merupakan metode pemberian halaman secara maju (*forward mapped paging*). Pemberian halaman dengan cara ini menggunakan pembagian tingkat setiap segmen alamat logikal. Setiap segmen menunjukkan indeks dari tabel halaman, kecuali segmen terakhir yang menunjukkan langsung frame pada memori fisik. Segmen terakhir ini biasa disebut offset(D). Dapat disimpulkan bahwa segmen yang terdapat dalam alamat logik menentukan berapa lapis paging yang digunakan yaitu banyak segmen-1. Selain itu dalam sistem ini setiap tabel halaman memiliki page table lapis kedua yang berbeda.

Dengan metoda ini, isi pada indeks tabel halaman pertama akan menunjuk pada tabel halaman kedua yang bersesuaian dengan isi dari tabel halaman pertama tersebut. Sedangkan isi dari page table kedua menunjukkan tempat di mana tabel halaman ketiga bermula, sedang segmen alamat logik kedua adalah indeks ke-n setelah starting point tabel halaman ketiga dan seterusnya sampai dengan segmen terakhir. Sedangkan segmen terakhir menunjukkan langsung indeks setelah alamat yang ditunjukkan oleh tabel halaman terakhir.



Gambar 12 Tabel halaman secara bertingkat

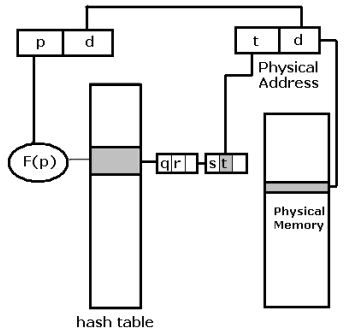
Kekurangan dari metoda ini adalah borosnya memori karena setiap tabel halaman menunjuk pada page tabel lainnya yang berbeda. Jika segmen pertama berisi p bit, maka besar tabel halaman pertama sebesar 2 pangkat p. Bila segmen kedua berisi s bit, maka setiap tabel halaman pertama menunjuk pada 2 pangkat s banyaknya ruang pada memori. Sehingga sampai dengan tingkat dua ini ruang yang dibutuhkan untuk paging sudah mencapai 2 pangkat s+p. Dapat dikatakan bahwa metoda ini sangat tidak cocok untuk diterapkan pada mapping besar seperti 64-bit walaupun dapat saja dilakukan.



Gambar 13 Penghalaman hirarki

Tabel Halaman Secara H*ashed*

Tabel Halaman secara Hashed cukup cocok untuk paging berukuran besar, seperti 64-bit. Karakteristik dari metoda ini adalah digunakannya sebuah fungsi untuk memanipulasi alamat logik. Selain sebuah fungsi, tabel halaman secara hashed juga menggunakan tabel hash dan juga pointer untuk menangani linked list. Hasil dari hashing akan dipetakan pada hash tabel halaman yang berisi linked list. Penggunaan linked list adalah untuk pengacakan data yang dikarenakan besar hash table yang sangat terbatas. Pada metoda ini offset masih sangat berperan untuk menunjukkan alamat fisik, yaitu dengan menggabungkan isi dari linked list dengan offset tersebut. Sistem ini dapat dikembangkan menjadi *Clustered* Tabel Halaman yang lebih acak dalam pengalamatan dalam memori fisik.

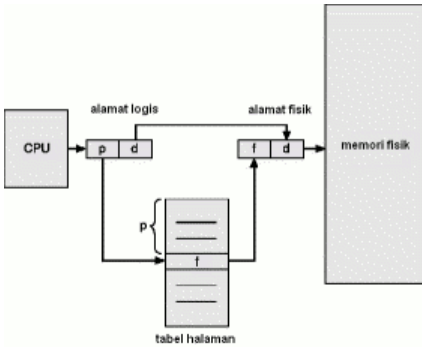


Gambar 14 Hashed page tables

Tabel Halaman Secara *Inverted*

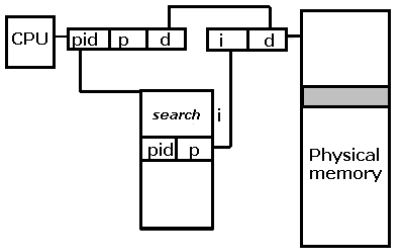
Metoda ini berbeda dengan metode lainnya. Pada Tabel Halaman Inverted, proses pemberian halaman dipusatkan pada proses yang sedang ditangani. Alamat logika yang menggunakan *inverted* tabel halaman merepresentasikan proses yang dimiliki. Sehingga tabel halaman pada metoda ini sama besar atau lebih dengan jumlah proses yang dapat ditangani dalam setiap kesempatan. Bila diasumsikan sistem dapat menangani n buah proses maka paling tidak tabel halaman juga sebesar n, sehingga setidaknya satu proses memiliki satu halaman yang bersesuaian dengan page tersebut.

Metoda inverted ini bertumpu pada proses pencarian identitas dari proses di dalam tabel halaman tersebut. Jika proses sudah dapat ditemukan di dalam tabel halaman maka letak indeks di tabel halaman yang dikirimkan dengan offset sehingga membentuk alamat fisik yang baru. Karena *Inverted paging* membatasi diri pada banyaknya proses maka jika dibandingkan dengan *hierarchical paging* metoda ini membutuhkan memori yang lebih sedikit.



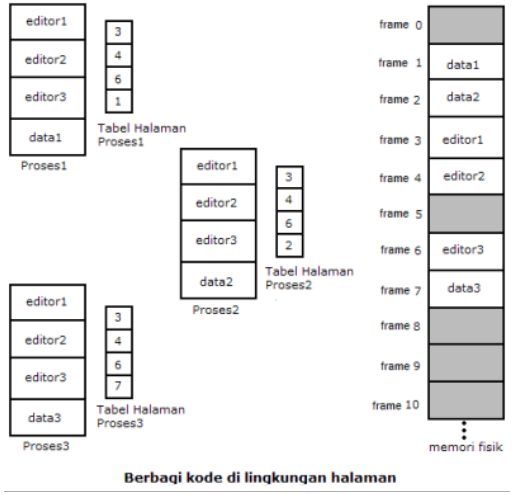
Gambar 15 Tabel halaman secara inverted

Kekurangan pertama dari inverted paging ini disebabkan oleh fasilitas searching yang dimilikinya. Memori fisik sendiri tersusun secara berurutan di dalam tabel halaman, namun proses yang ingin dicari berasal dari alamat virtual sehingga ada kemungkinan dilakukan pencarian seluruh tabel halaman sampai akhirnya halaman tersebut ditemukan. Hal ini menyebabkan ketidakstabilan metoda ini sendiri. Kekurangan lain dari inverted tabel halaman adalah sulitnya menerapkan memori berbagi (*shared*). Memori sharing adalah sebuah cara dimana proses yang berbeda dapat mengakses suatu alamat di memori yang sama. Ini bertentangan dengan konsep dari inverted tabel halaman sendiri yaitu, setiap proses memiliki satu atau lebih frame di memori, dengan pasangan proses dan frame unik. Unik dalam arti beberapa frame hanya dapat diakses oleh satu buah proses saja. Karena perbedaan konsep yang sedemikian jauh tersebut maka memori sharing hampir mustahil diterapkan dengan inverted tabel halaman.



Gambar 16 Inverted page tables

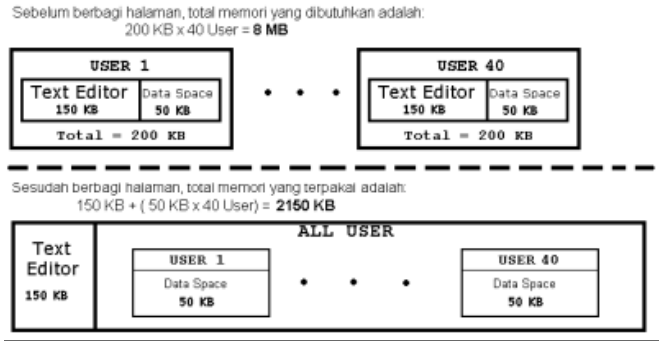
Berbagi Halaman (Share)



Gambar 1.17 Berbagi halaman

Keuntungan lain dari pemberian halaman adalah kemungkinannya untuk berbagi kode yang sama. Pertimbangan ini terutama sekali penting pada lingkungan yang berbagi waktu. Pertimbangkan sebuah sistem yang mendukung 40 pengguna, yang masing-masing menjalankan aplikasi pengedit teks. Jika editor teks tadi terdiri atas 150K kode dan 50K ruang data, kita akan membutuhkan 8000K untuk mendukung 40 pengguna. Jika kodenya dimasukan ulang, bagaimana pun juga dapat dibagi-bagi, seperti pada gambar. Disini kita lihat bahwa tiga halaman editor (masing-masing berukuran 50K; halaman ukuran besar digunakan untuk menyederhanakan gambar) sedang dibagi-bagi diantara tiga proses. Masing-masing proses mempunyai halaman datanya sendiri. Terlihat jelas selisih penggunaan memori sesudah dan sebelum berbagi halaman adalah sebesar 5850MB. Frame yang berisi editor

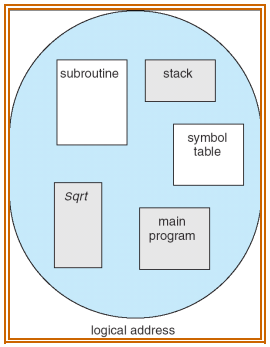
diakses oleh banyak pengguna. Jadi hanya ada satu salinan editor yang ditaruh di memori. Tiap tabel halaman dari tiap proses mengakses editor yang sama, tapi halaman data dari tiap proses tetap ditaruh di frame yang berbeda. Inilah yang dimaksud dengan berbagi halaman.



Gambar 18 Share page

Segmentasi

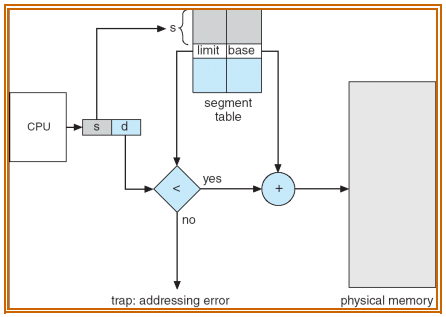
Segmentasi merupakan skema manajemen memori yang mendukung cara pandang seorang programmer terhadap memori. Ruang alamat logika merupakan sekumpulan dari segmen-segmen. Masing-masing segment mempunyai panjang dan nama. Alamat diartikan sebagai nama segmen dan offset dalam suatu segmen. Jadi jika seorang pengguna ingin menunjuk sebuah alamat dapat dilakukan dengan menunjuk nama segmen dan offsetnya. Untuk lebih menyederhanakan implementasi, segmen-segmen diberi nomor yang digunakan sebagai pengganti nama segment. Sehingga, alamat logika terdiri dari dua tupple: [segment-number, offset].



Gambar 19 Sisi pandang program oleh pengguna

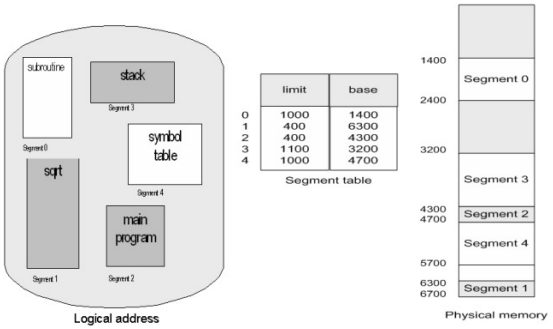
Segmentasi Perangkat Keras

Meskipun seorang pengguna dapat memandang suatu objek dalam suatu program sebagai alamat berdimensi dua, memori fisik yang sebenarnya tentu saja masih satu dimensi barisan byte. Jadi kita harus bisa mendefinisikan pemetaan dari dua dimensi alamat yang didefinisikan oleh pengguna ke satu dimensi alamat fisik. Pemetaan ini disebut sebagai sebuah segment table. Masing-masing masukan dari mempunyai segment base dan segment limit. Segment base merupakan alamat fisik dan segmen limit diartikan sebagai panjang dari segmen.



Gambar 20 Arsitektur segmentasi perangkat keras

Suatu alamat logika terdiri dari dua bagian, yaitu nomor segmen(s), dan offset pada segmen(d). Nomor segmen digunakan sebagai indeks dalam segmen table. Offset d alamat logika harus antara 0 hingga dengan segmen limit. Jika tidak maka diberikan pada sistem operasi. Jika offset ini legal maka akan dijumlahkan dengan segmen base untuk menjadikannya suatu alamat di memori fisik dari byte yang diinginkan. Jadi segmen table ini merupakan suatu array dari pasangan base dan limit register.

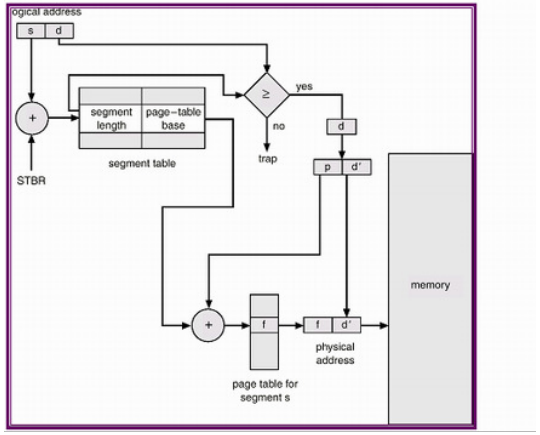


Gambar 21 Segmentasi

Sebagai contoh kita mempunyai nomor segmen dari 0 sampai dengan 4. Segmen-segmen ini disimpan dalam suatu memori fisik. Tabel segmen berisi data untuk masing-masing segmen, yang memberikan informasi tentang awal alamat dari segmen di fisik memori (atau base) dan panjang dari segmen (atau limit). Misalkan, segmen 2 mempunyai panjang 400 dan dimulai pada lokasi 4300. Jadi, referensi di byte 53 dari segmen 2 dipetakan ke lokasi 4300 + 53 = 4353. Suatu referensi ke segmen 3, byte 852, dipetakan ke 3200 (sebagai base dari segmen) + 852 = 4052. Referensi ke byte 1222 dari segmen 0 akan menghasilkan suatu trap ke sistem operasi, karena segmen ini hanya mempunyai panjang 1000 byte.

Keuntungan Segmentasi

Kelebihan Pemberian Halaman: tidak ada fragmentasi luar-alokasinya cepat.
Kelebihan Segmentasi: saling berbagi-proteksi.



Gambar 22 Segmentasi dengan pemberian halaman

Keuntungan pemakaian cara segmentasi ini adalah sebagai berikut:

1. Menyederhanakan penanganan struktur data yang berkembang. Seringkali penanganan struktur data menuntut perubahan panjang data. Hal ini dimungkinkan dengan adanya segmentasi. Jadi dengan segmentasi membuat penanganan struktur data menjadi fleksibel.

2. Kompilasi ulang independen tanpa mentautkan kembali seluruh program. Teknik ini memungkinkan program-program dikompilasi ulang secara independen tanpa perlu mentautkan kembali seluruh program dan dimuatkan kembali. Jika masing-masing prosedur terdapat di segmen terpisah beralamat 0 sebagai alamat awal, maka pentautan prosedur-prosedur yang dikompilasi secara terpisah sangat lebih mudah. Setelah semua prosedur dikompilasi dan ditautkan, panggilan ke prosedur di segmen n akan menggunakan alamat dua bagian yaitu (n,0) mengacu ke word alamat 0 (sebagai titik masuk) segmen ke n. Jika prosedur di segmen n dimodifikasi dan dikompilasi ulang, prosedur lain tidak perlu diubah (karena tidak ada modifikasi alamat awal) walau versi baru lebih besar dibanding versi lama.

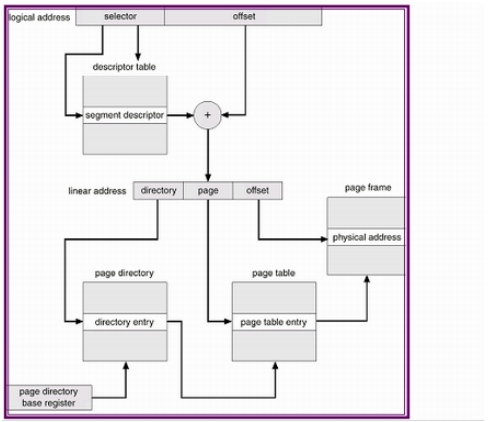
3. Memudahkan pemakaian memori bersama diantara proses-proses Teknik ini memudahkan pemakaian memori bersama diantara proses-proses. Pemrogram dapat menempatkan program utilitas atau tabel data berguna di segmen yang dapat diacu oleh proses-proses lain. Segmentasi memberi fasilitas pemakaian bersama terhadap prosedur dan data untuk dapat diproses, berupa shared library.

   Pada workstation modern yang menjalankan sistem Windows sering mempunyai pustaka grafis sangat besar. Pustaka ini diacu hampir semua program. Pada sistem bersegmen, pustaka grafis diletakan di satu segmen dan dipakai secara bersama banyak proses sehingga menghilangkan mempunyai pustaka ditiap ruang alamat proses. Shared libraries di sistem pengalamantan murni lebih rumit, yaitu dengan simulasi segmentasi.

4. Memudahkan proteksi karena segmen dapat dikonstruksi berisi sekumpulan prosedur atau data terdefinisi baik, pemrogram atau administrator sistem dapat memberikan kewenangan pengaksesan secara nyaman.

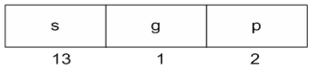
Penggunaan Segmentasi pada Pentium

Arsitektur Pentium memperbolehkan segmen sebanyak 4 GB dan jumlah maksimum segmen per proses adalah 16 KB. Ruang alamat logika dari proses dibagi menjadi 2 partisi. Partisi pertama terdiri atas segmen-segmen hingga 8 KB (tersendiri dari proses). Partisi kedua terdiri atas segmen-segmen hingga 8 KB yang berbagi dengan semua proses-proses. Informasi partisi pertama terletak di LDT (Local Descriptor Table), informasi partisi kedua terletak di GDT (Global Descriptor Table).



Gambar 23 Segmentasi dengan pemberian halaman (INTEL 30386)

Masing-masing entri dari LDT dan GDT terdiri atas 8 byte segmen descriptor dengan informasi yang rinci tentang segmen-segmen tertentu, termasuk lokasi base dan limit dari segmen itu. Alamat logikal adalah sepasang (selector, offset), dimana berjumlah 16 bit. Gambarnya adalah sebagai berikut:

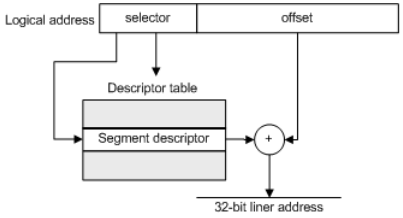


Gambar 24 Selector

Dimana s menandakan nomor segmen, g mengindikasikan apakah segmen GDT atau LDT, dan p mengenai proteksi. Offsetnya berjumlah 32 bit yang menspesifikasi lokasi byte (atau word) dalam segmentasi Pentium.

Pada Pentium mempunyai 6 register mikroprogram 8 byte, yang mengijinkan 6 segmen tadi untuk dialamatkan kapan saja. 6 register ini berfungsi untuk menangani deskriptor-deskriptor yang sesuai dengan LDT atau GDT. Cache ini juga mengijinkan Pentium untuk tidak membaca deskriptor dari memori.

Alamat linear pada Pentium panjangnya 32 bit dan prosesnya adalah register segmen menunjuk pada entry yang sesuai dalam LDT atau GDT. Informasi base dan limit tentang segmen Pentium digunakan untuk menghasilkan alamat linear. Pertama, limit digunakan untuk memeriksa valid tidaknya suatu alamat. Jika alamat tidak valid, maka kesalahan memori akan terjadi yang menimbulkan trap pada sistem operasi. Jika alamat valid, maka nilai offset dijumlahkan dengan nilai base, yang menghasilkan alamat linear 32 bit. Hal ini ditunjukkan seperti pada gambar berikut:



Gambar 25 Segmentasi Intel Pentium

Segmentasi pada Linux

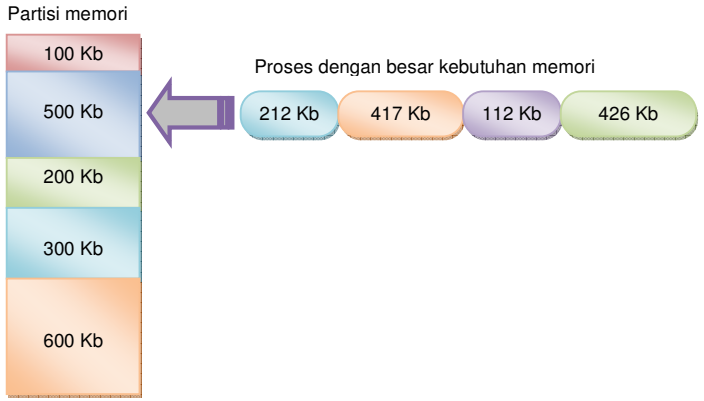
Pada Pentium, Linux hanya menggunakan 6 segmen:
1. Segmen untuk kode kernel.
2. Segmen untuk data kernel.
3. Segmen untuk kode pengguna.
4. Segmen untuk data pengguna.
5. Segmen Task-state (TSS).
6. Segmen default LDT.

Segmen untuk kode pengguna dan data pengguna berbagi dengan semua proses yang running pada pengguna mode, karena semua proses menggunakan ruang alamat logika yang sama dan semua descriptor segmen terletak di GDT. TSS (Task-state Segment) digunakan untuk menyimpan context hardware dari tiap proses selama context switch. Tiap proses mempunyai TSS sendiri, dimana deskriptornya terletak di GDT. Segmen default LDT normalnya berbagi dengan semua proses dan biasanya tidak digunakan. Jika suatu proses membutuhkan LDT-nya, maka proses dapat membuatnya dan tidak menggunakan default LDT. Seperti yang telah dijelaskan, tiap selektor segmen mempunyai 2 bit proteksi. Maka, Pentium mengijinkan proteksi 4 level. Dari 4 level ini, Linux hanya mengenal 2 level, yaitu pengguna mode dan kernel mode.

DISKUSI

1. Jelaskan perbedaan antara fragmentasi internal dan eksternal, dan berikan contohnya!

2. Diberikan 5 (lima) partisi dengan ukuran masing-masing secara terurut adalah 100Kb, 500Kb, 200Kb, 300Kb, dan 600Kb. Jelaskan bagaimana tiap-tiap algoritma first-fit, best-fit dan worst-fit menempatkan proses dengan ukuran memori masing-masing sebesar 212Kb, 417Kb, 112 Kb, dan 426Kb (secara berurut)! Algoritma mana yang lebih efisien penggunaan memorinya?



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