MODUL SISTEM OPERASI

(CCI210)

MODUL SESI 6

MEMORY MANAGEMENT

DISUSUN OLEH

ADI WIDIANTONO, SKOM, MKOM.

UNIVERSITAS ESA UNGGUL

2020

MEMORY MANAGEMENT

A. Kemampuan Akhir Yang Diharapkan

Setelah mempelajari modul ini, diharapkan mahasiswa mampu :

1. Memahami konsep dasar dari memory management.
2. Memahami teknik dari manajemen memory.
3. Mamahami proses kerja Sistem Operasi dalam menjalankan manajemen memory.

B. Uraian dan Contoh

1. Konsep Manajemen Memori

Memori

Memori sebagai tempat penyimpanan instruksi/ data dari program. Memori adalah pusat kegiatan pada sebuah komputer, karena setiap proses yang akan dijalankan, harus melalui memori terlebih dahulu. Untuk dapat dieksekusi, program harus dibawa ke memori dan menjadi suatu proses.

Manajemen memori

Manajemen memori meliputi pengaturan dasar seperti:

- Melacak pemakaian memori (siapa dan berapa besar)
- Memilih program mana yang akan diload ke memori
- Alokasi dan dealokasi memori fisik untuk program/ proses-proses dalam menggunakan address space

Tugas Sistem Operasi

Dalam hal ini tugas Sistem Operasi menggunakan menejemen memori untuk mengatur peletakan banyak proses pada suatu memori, memori harus dapat digunakan dengan baik → dapat memuat banyak proses dalam suatu waktu.

Konsep Dasar Manajemen Memori

Konsep dasar manajemen memori meliputi pemahaman dari:

1. Konsep dasar memori
    a. Konsep Binding
    b. Dynamic Loading
    c. Dynamic Linking
    d. Overlay
2. Ruang Alamat Logika dan Fisik
3. Swapping
4. Pengalokasian Berurutan (Contiguous Allocation)
5. Pengalokasian Tidak Berurutan (Non Contiguous Allocation)

2. Kebutuhan Manajemen Memori

Saat mempelajari berbagai mekanisme dan kebijakan yang terkait dengan manajemen memori, ada baiknya untuk mengetahui kebutuhan yang akan dipenuhi oleh manajemen memori. Kebutuhan tersebut meliputi:

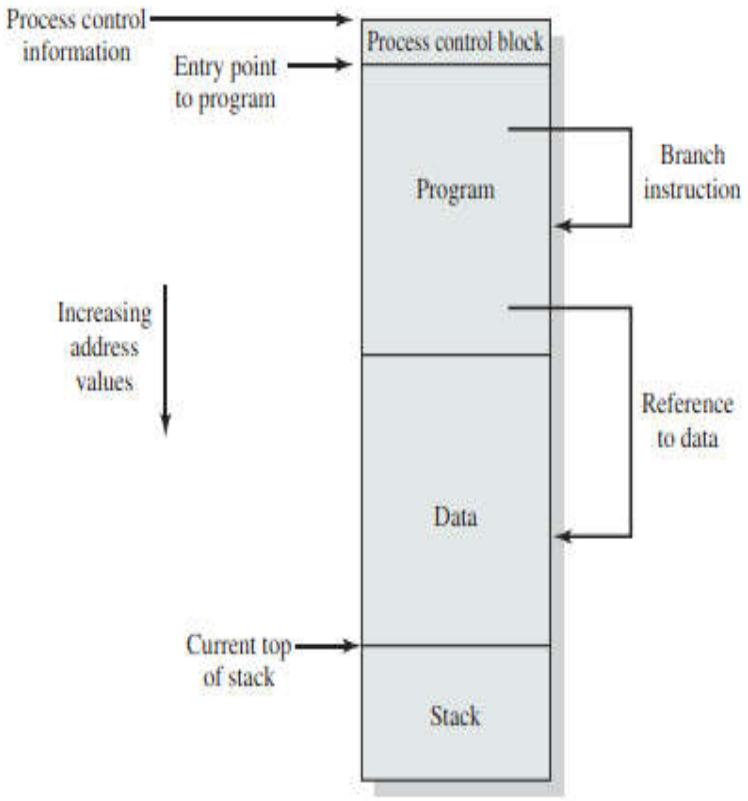
- Relokasi
- Perlindungan
- Berbagi
- Organisasi logis
- Organisasi fisik

Relokasi (Relocate)

Dalam sistem multiprogramming, memori utama yang tersedia biasanya digunakan bersama oleh sejumlah proses. Pada umumnya, programmer tidak mungkin mengetahui lebih dahulu program mana yang akan disimpan di memori utama pada saat pelaksanaan programnya. Selain itu, diinginkan untuk menukar proses aktif keluar dan masuk dari memori utama untuk memaksimalkan pemanfaatan prosesor dalam menyediakan kumpulan proses dalam jumlah besar yang siap untuk dieksekusi. Setelah program dikeluarkan ke disk, akan sangat membatasi untuk dimasukan kembali, untuk menempati wilayah memori utama yang sama seperti sebelumnya. Dan saat itu mungkin dibutuhkan untuk memindahkan/**merelokasi** (**relocate**) proses ke area memori yang berbeda.

Jadi, kita tidak bisa tahu sebelumnya di mana program akan ditempatkan, dan harus memungkinkan program dapat dipindahkan di memori utama karena bertukar. Fakta-fakta ini menimbulkan beberapa masalah teknis terkait dengan menangani, seperti yang diilustrasikan pada Gambar 2.1. Gambar tersebut menggambarkan gambar proses. Sistem operasi perlu mengetahui lokasi kontrol proses informasi dan tumpukan eksekusi, serta titik masuk untuk memulai eksekusi program untuk proses ini. Karena sistem operasi mengelola memori dan bertanggung jawab untuk membawa proses ini ke dalam memori utama, alamat ini mudah didapat. Selain itu, bagaimanapun, prosesor harus berurusan dengan memori referensi dalam program. Instruksi cabang berisi alamat untuk referensi instruksi yang akan dijalankan selanjutnya. Instruksi referensi data berisi alamat dari *byte* atau data *word* yang direferensikan. Perangkat keras prosesor dan perangkat lunak sistem operasi harus dapat menerjemahkan referensi memori yang terdapat di file kode program ke

alamat memori fisik aktual, yang mencerminkan arus lokasi program di memori utama



Gambar 2.1. Addressing requirements for a process

Perlindungan (Protection)

Setiap proses harus dilindungi dari gangguan yang tidak diinginkan oleh proses lain, baik tidak disengaja atau disengaja. Jadi, program dalam proses lain seharusnya tidak dapat merujuk lokasi memori dalam tujuan proses membaca atau menulis tanpa izin. Di satu sisi, pemenuhan persyaratan relokasi meningkatkan kesulitan untuk memenuhi persyaratan perlindungan. Karena lokasi program di memori utama tidak dapat diprediksi, tidak mungkin periksa alamat absolut pada waktu kompilasi untuk memastikan perlindungan. Selain itu, sebagian besar bahasa pemrograman memungkinkan kalkulasi alamat secara dinamis pada saat dijalankan (misalnya, dengan menghitung subskrip array atau penunjuk ke dalam struktur data). Karena itu semuanya referensi memori yang dihasilkan oleh suatu proses harus diperiksa pada waktu berjalan untuk memastikannya bahwa mereka hanya mengacu

pada ruang memori yang dialokasikan untuk proses itu. Untungnya mekanisme yang mendukung relokasi juga mendukung persyaratan perlindungan.

Biasanya, proses pengguna tidak dapat mengakses bagian mana pun dari sistem operasi, baik program maupun data. Sekali lagi, biasanya program dalam satu proses tidak dapat bercabang ke instruksi dalam proses lain. Tanpa pengaturan khusus, program menjadi satu proses tidak dapat mengakses area data proses lain. Prosesor harus mampu untuk membatalkan instruksi tersebut pada saat eksekusi.

Persyaratan perlindungan memori harus dipenuhi oleh prosesor (perangkat keras) daripada sistem operasi (perangkat lunak). Hal ini dikarenakan OS tidak dapat mengantisipasi semua referensi memori yang akan dibuat oleh sebuah program.

Berbagi (Shared)

Mekanisme perlindungan apa pun harus memiliki fleksibilitas untuk memungkinkan beberapa proses mengakses bagian yang sama dari memori utama. Misalnya, jika sejumlah proses menjalankan program yang sama, adalah lebih mudah untuk mengizinkan setiap proses mengakses file salinan program yang sama daripada memiliki salinan terpisahnya sendiri. Memproses itu bekerja sama dalam beberapa tugas mungkin perlu berbagi akses ke struktur data yang sama. Oleh karena itu, sistem manajemen memori harus mengizinkan akses terkontrol untuk membagikan (***sharing***) area memori tanpa mengorbankan perlindungan penting.

Organisasi Logis

Hampir selalu, memori utama dalam sistem komputer diatur sebagai linier, atau satu dimensi, ruang alamat, terdiri dari urutan *byte* atau *word*. Memori sekunder, pada tingkat fisiknya, diatur dengan cara yang sama. Meskipun organisasi ini sangat mirip dengan perangkat keras mesin yang sebenarnya, namun tidak sesuai dengan cara program dibangun. Sebagian besar program diatur menjadi modul, beberapa di antaranya tidak dapat dimodifikasi (hanya baca, hanya eksekusi) dan beberapa di antaranya yang berisi data yang dapat dimodifikasi. Jika sistem operasi dan perangkat keras komputer dapat secara efektif menangani program pengguna dan data dalam bentuk modul dari beberapa jenis, maka terdapat keuntungan dapat direalisasikan:

1. Modul dapat ditulis dan disusun secara independen, dengan semua referensi dari satu modul ke modul lainnya diselesaikan oleh sistem pada saat berjalan.

2. Dengan beban tambahan yang sederhana, tingkat perlindungan yang berbeda (hanya baca, mengeksekusi saja) dapat diberikan ke modul yang berbeda.
3. Dimungkinkan untuk memperkenalkan mekanisme dimana modul dapat dibagi proses. Keuntungan dari menyediakan berbagi di tingkat modul adalah sesuai dengan cara pengguna melihat masalah, dan karenanya mudah untuk dilakukan pengguna untuk menentukan pembagian yang diinginkan.

Metode yang paling siap memenuhi kebutuhan ini adalah segmentasi, merupakan salah satu teknik manajemen memori (akan dibahas pada materi berikutnya).

Organisasi Fisik

Secara umum, memori komputer diatur menjadi setidaknya dua level, disebut sebagai memori utama dan memori sekunder. Memori utama menyediakan akses cepat dengan beban yang relatif tinggi. Selain itu, memori utama mudah berubah; tidak menyediakan penyimpanan permanen. Memori sekunder lebih lambat dan lebih murah daripada memori utama dan biasanya tidak mudah hilang. Dengan demikian memori sekunder berkapasitas besar dapat disediakan untuk penyimpanan program dan data jangka panjang, sementara utama yang lebih kecil memori menyimpan program dan data yang sedang digunakan.

Dalam skema dua tingkat ini, organisasi dari aliran informasi antara memori utama dan sekunder merupakan perhatian sistem utama. Tanggung jawab untuk aliran ini dapat ditugaskan ke programmer individu, namun hal ini tidak praktis dan tidak diinginkan karena dua alasan:

1. Memori utama yang tersedia untuk sebuah program ditambah datanya mungkin tidak cukup. Di dalam hal ini, programmer harus terlibat dalam praktik yang dikenal sebagai ***overlay***, di dimana program dan data diorganisasikan sedemikian rupa sehingga berbagai modul dapat diberikan wilayah memori yang sama, dengan program utama yang bertanggung jawab untuk mengganti modul masuk dan keluar sesuai kebutuhan.
2. Dalam lingkungan multiprogramming, programmer tidak mengetahui di waktu pengkodean berapa banyak ruang yang akan tersedia atau di mana ruang itu akan berada. Jelaslah, bahwa tugas memindahkan informasi antara dua tingkat memori harus menjadi tanggung jawab sistem. Tugas ini adalah inti dari manajemen memori.

3. TIPE PENGALOKASIAN MEMORI

Contiguous Allocation

Pada Multiprogramming memori utama harus mengalokasikan tempat untuk sistem operasi dan beberapa user proses. Memori harus mengakomodasi baik OS dan proses user. Memori dibagi menjadi 2 partisi :

- Untuk OS yang resident
- Untuk Proses User

Ada 2 tipe Contiguos Allocation :

- Single Partition (Partisi Tunggal)
- Multiple Partition (Partisi Banyak)

Non Contiguous Allocation

Paging

- Paging adalah solusi untuk permasalahan fragmentasi external
- Memori fisik dibagi ke dalam blok-blok ukuran tetap yang disebut “frame”
- Memori logika dibagi ke dalam blok-blok dengan ukuran yang sama yang disebut “page”
- Untuk menjalankan program berukuran n page, harus dicari frame kosong sebanyak n untuk meload program
- Page table digunakan untuk translasikan alamat lojik ke alamat fisik

Mengenai Paging akan dibahas khusus di materi berikutnya.

4. Teknik Manajemen Memori

Beberapa teknik manajemen memori dapat dilihat dalam tabel berikut ini:

Technique	Description	Strengths	Weaknesses
Fixed Partitioning	Main memory is divided into a number of static partitions at system generation time. A process may be loaded into a partition of equal or greater size.	Simple to implement; little operating system overhead.	Inefficient use of memory due to internal fragmentation; maximum number of active processes is fixed.
Dynamic Partitioning	Partitions are created dynamically, so that each process is loaded into a partition of exactly the same size as that process.	No internal fragmentation; more efficient use of main memory.	Inefficient use of processor due to the need for compaction to counter external fragmentation.
Simple Paging	Main memory is divided into a number of equal-size frames. Each process is divided into a number of equal-size pages of the same length as frames. A process is loaded by loading all of its pages into available, not necessarily contiguous, frames.	No external fragmentation.	A small amount of internal fragmentation.
Simple Segmentation	Each process is divided into a number of segments. A process is loaded by loading all of its segments into dynamic partitions that need not be contiguous.	No internal fragmentation; improved memory utilization and reduced overhead compared to dynamic partitioning.	External fragmentation.
Virtual Memory Paging	As with simple paging, except that it is not necessary to load all of the pages of a process. Nonresident pages that are needed are brought in later automatically.	No external fragmentation; higher degree of multiprogramming; large virtual address space.	Overhead of complex memory management.
Virtual Memory Segmentation	As with simple segmentation, except that it is not necessary to load all of the segments of a process. Nonresident segments that are needed are brought in later automatically.	No internal fragmentation, higher degree of multiprogramming; large virtual address space; protection and sharing support.	Overhead of complex memory management.

Dalam materi ini akan dibahas mengenai teknik yang berkaitan dengan teknik ***partitioning***. Segmentasi dan paging akan dibahas pada materi berikutnya.

4.1. Memory Partitioning (Partisi Memori)

4.1.1. Partisi Tetap (Fix Partitioning)

Dalam kebanyakan skema untuk manajemen memori, kita dapat mengasumsikan bahwa OS menempati beberapa bagian tetap dari memori utama dan sisa memori utama tersedia untuk digunakan oleh banyak proses. Skema paling sederhana untuk mengelola ini tersedia memori adalah untuk mempartisi menjadi wilayah dengan batas tetap.

UKURAN PARTISI (partition sizes) Gambar 4.2 menunjukkan contoh dua alternatif untuk partisi tetap. Salah satu kemungkinannya adalah dengan menggunakan partisi berukuran sama. Pada kasus ini, proses apa pun yang ukurannya kurang dari atau sama dengan ukuran partisi dapat dimuat partisi yang tersedia. Jika semua partisi sudah penuh dan tidak ada proses dalam status *Ready* atau *Running*, sistem operasi dapat menukar proses dari salah satu partisi dan memuat proses lain, sehingga ada beberapa pekerjaan untuk prosesor.

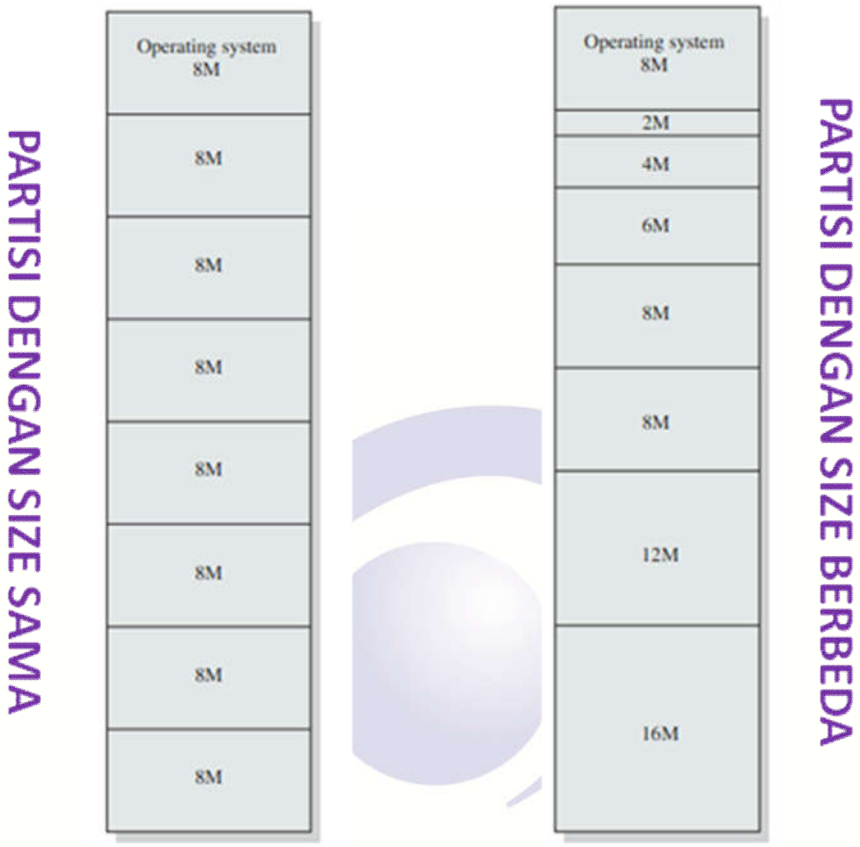
Ada dua kesulitan dengan penggunaan partisi tetap berukuran sama:

• Sebuah program mungkin terlalu besar untuk dimasukkan ke dalam partisi. Dalam hal ini, programmer harus mendesain program dengan menggunakan overlay sehingga hanya sebagian saja Program harus berada di memori utama pada satu waktu. Saat modul dibutuhkan yang tidak ada, program pengguna harus memuat modul itu ke dalam partisi program, meng-overlay program atau data apa pun yang ada di sana.

• Pemanfaatan memori utama sangat tidak efisien. Program apa pun, tidak masalah seberapa kecil, menempati seluruh partisi. Dalam contoh kita, mungkin ada program yang panjangnya kurang dari 2 Mbytes; namun itu menempati partisi 8-Mbyte Fenomena ini, di mana ada ruang yang terbuang internal ke partisi karena fakta bahwa blok data yang dimuat lebih kecil daripada partisi, hal disebut sebagai **fragmentasi internal**.

Kedua masalah ini dapat dikurangi, meskipun tidak dapat diselesaikan, dengan menggunakan partisi yang tidak sama (Gambar 4.2b). Dalam contoh ini, program sebesar 16 Mbytes bisa ditampung tanpa overlay. Partisi yang lebih kecil dari 8

Mbytes memungkinkan lebih kecil program untuk diakomodasi dengan lebih sedikit fragmentasi internal.



Gambar 4.2 (a)(b)

ALGORITMA PENEMPATAN Dengan partisi berukuran sama, penempatan proses dalam memori menjadi hal yang mudah. Selama ada partisi yang tersedia, suatu proses dapat dilakukan dimuat ke partisi itu. Jika semua partisi ditempati dengan proses yang tidak siap dijalankan, maka salah satu dari proses ini harus ditukar untuk memberi ruang bagi sebuah proses baru.

Dengan ukuran partisi yang tidak sama, ada dua cara yang mungkin untuk menetapkan proses ke partisi. Cara termudah adalah menetapkan setiap proses ke partisi terkecil yang muat. Dalam hal ini, antrian penjadwalan diperlukan untuk setiap partisi, untuk menahan proses pertukaran yang ditujukan untuk partisi tersebut (Gambar 4.3a). Keuntungan dari pendekatan ini adalah bahwa proses selalu ditetapkan sedemikian rupa meminimalkan memori yang terbuang dalam partisi (**fragmentasi internal**).

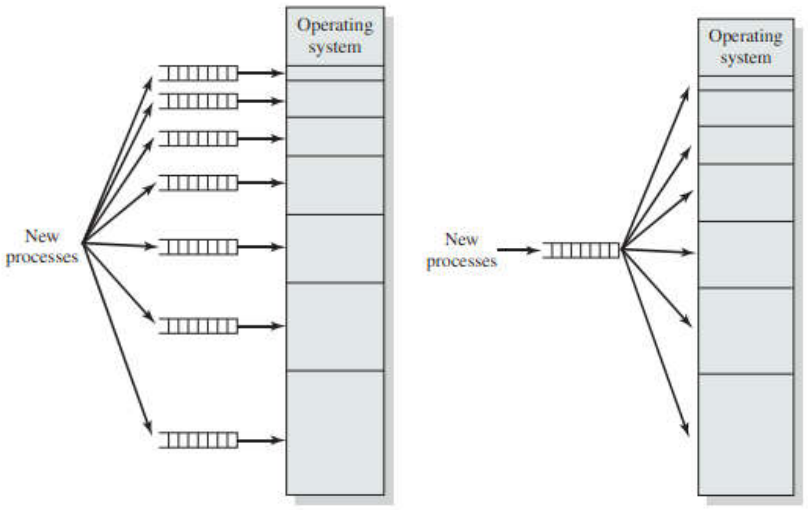
Tteknik ini tampak optimal dari sudut pandang partisi individu, namun tidak optimal dari sudut pandang sistem secara keseluruhan. Pada Gambar 4.2b, misalnya, pertimbangkan kasus di mana tidak ada proses dengan ukuran antara 12 dan 16M pada titik waktu tertentu, partisi 16M akan tetap tidak digunakan, meskipun beberapa proses yang lebih kecil mengantri. Dengan demikian, pendekatan yang lebih disukai adalah menggunakan satu antrian untuk semua proses (Gambar 4.3b). Ketika tiba waktunya untuk memuat sebuah proses ke dalam memori utama, yang terkecil partisi yang tersedia yang akan menahan proses dipilih. Jika semua partisi sudah terisi, maka keputusan pertukaran harus dibuat. Preferensi mungkin diberikan untuk bertukar dari partisi terkecil yang akan menampung proses masuk. Dapat juga mempertimbangkan faktor lain, seperti prioritas, dan preferensi untuk menukar proses diblokir versus proses yang siap.

Penggunaan partisi dengan ukuran yang tidak sama memberikan tingkat fleksibilitas untuk diperbaiki partisi. Selain itu, dapat dikatakan bahwa skema partisi tetap relative sederhana dan memerlukan perangkat lunak OS serta biaya pemrosesan yang minimal. Namun, ada kekurangannya yaitu:

• Jumlah partisi yang ditentukan pada waktu pembuatan sistem membatasi jumlah dari proses aktif (tidak ditangguhkan) dalam sistem.

• Karena ukuran partisi sudah diatur sebelumnya pada waktu pembuatan sistem, pekerjaan kecil tidak akan memanfaatkan ruang partisi secara efisien.

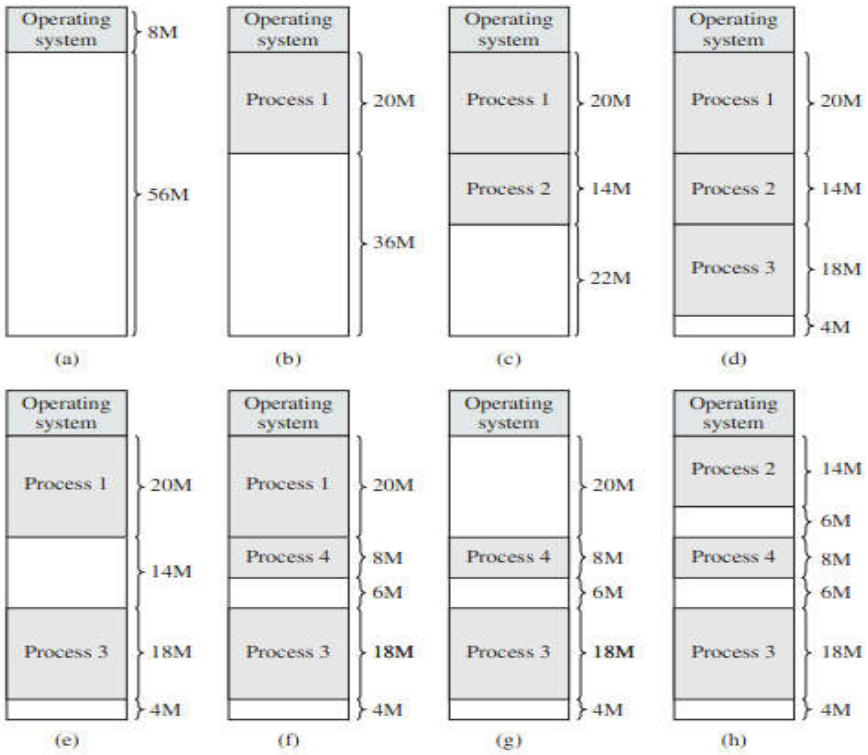
Penggunaan partisi tetap hampir tidak dikenal saat ini. Salah satu contoh sistem operasi yang sukses yang menggunakan teknik ini adalah sistem operasi mainframe IBM awal, dengan OS / MFT (Multiprogramming dengan Jumlah Tugas Tetap).



Gambar 4.3. (a) antrian setiap partisi (b) satu antrian untuk semua partisi

4.1.2. Partisi Dinamis (Dynamic Partitioning)

Untuk mengatasi beberapa kesulitan dengan partisi tetap, sebuah pendekatan dikenal sebagai partisi dinamis dikembangkan. Dengan partisi dinamis, partisi memiliki panjang dan jumlah variabel. Ketika sebuah proses dibawa ke memori utama, itu dialokasikan memori persis sebanyak yang dibutuhkan dan tidak lebih. Contoh, menggunakan 64 Mbytes memori utama, lihat Gambar 4.4. Awalnya, memory utama kosong, kecuali OS (a). Tiga proses pertama dimuat, dimulai dari akhir sistem operasi dan menempati ruang yang cukup untuk setiap proses (b, c, d). Dan meninggalkan "lubang" di akhir memori yang terlalu kecil untuk proses keempat. Pada titik tertentu, tidak ada proses dalam memori sudah siap. Sistem operasi menukar proses 2 (e), yang menyisakan ruang yang cukup untuk memuat proses baru, proses 4 (f). Karena proses 4 lebih kecil dari proses 2, lubang kecil lainnya dibuat. Pada satu titik di mana tidak ada proses di memori utama yang siap, tetapi proses 2, di Status Siap-Tangguhkan, tersedia. Karena ruang memori tidak cukup untuk proses 2, sistem operasi menukar proses 1 keluar (g) dan proses menukar 2 kembali (h).



Gambar 4.4.

Metode ini dimulai dengan baik, tetapi akhirnya mengarah ke sebuah situasi di mana ada banyak lubang kecil di memori. Seiring berjalannya waktu, memori menjadi semakin terfragmentasi, dan pemanfaatan memori menurun. Ini fenomena disebut sebagai **fragmentasi eksternal**, menunjukkan bahwa memori yang di luar semua partisi menjadi semakin terfragmentasi.

Salah satu teknik untuk mengatasi fragmentasi eksternal adalah **pemadatan** (***compaction***): Dari dari waktu ke waktu, OS menggeser proses sehingga berdekatan dan semuanya memori bebas digabungkan dalam satu blok. Misalnya, pada Gambar 2.4h, pemadatan akan menghasilkan blok memori bebas dengan panjang 16M. Ini mungkin cukup untuk dimuat dalam proses tambahan. Kesulitan dengan pemadatan adalah bahwa ini merupakan prosedur yang memakan waktu dan membuang waktu prosesor.

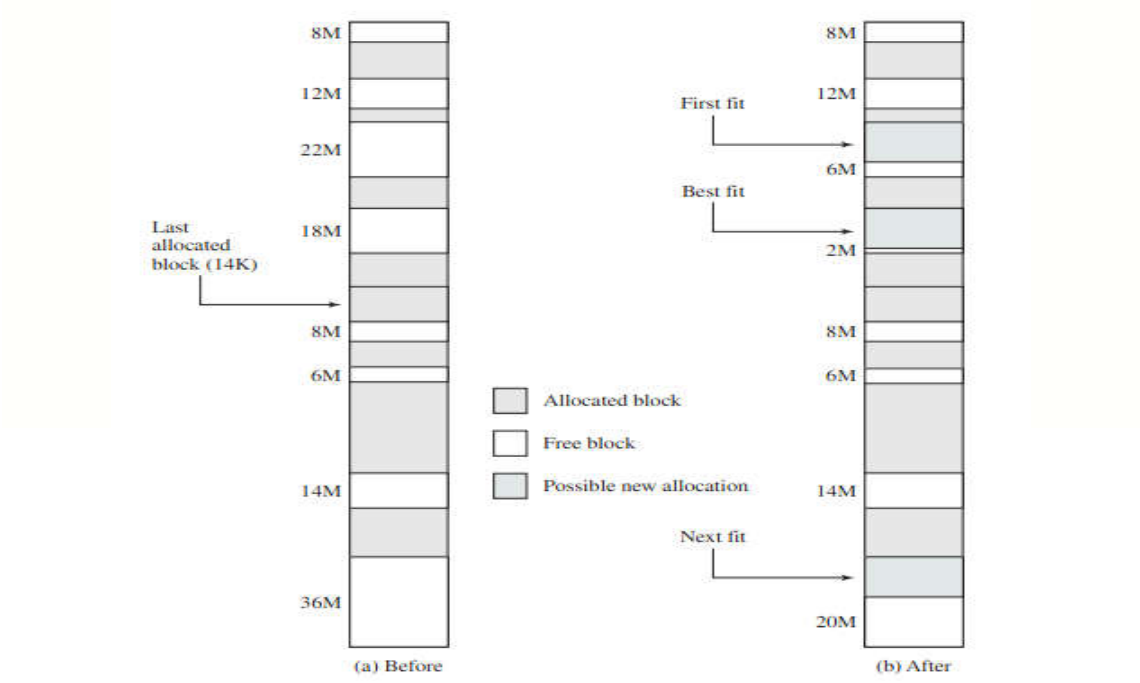
ALGORITMA PENEMPATAN Karena pemadatan memori memakan waktu, perancang OS harus pandai dalam memutuskan bagaimana menetapkan proses ke memori (cara menyambungkan lubang). Kapan saatnya untuk memuat atau menukar proses ke memori utama, dan jika ada lebih dari satu blok memori bebas dengan ukuran yang cukup, maka sistem operasi harus memutuskan blok bebas mana yang akan dialokasikan.

Tiga algoritme penempatan yang mungkin dapat dipertimbangkan adalah ***Best-Fit***, ***First-Fit***, dan ***Next-Fit***. Semua, terbatas pada memilih di antara blok memori utama yang bebas dengan ukuran yang sama atau lebih besar dari proses yang akan dibawa masuk. ***Best-Fit***,memilih blok yang ukurannya paling dekat dengan permintaan. ***First-fit*** mulai memindai memori dari memulai dan memilih blok pertama yang tersedia yang cukup besar. ***Next-Fit*** dimulai untuk memindai memori dari lokasi penempatan terakhir, dan memilih blok berikutnya yang tersedia yang cukup besar.

Gambar 4.5a menunjukkan contoh konfigurasi memori setelah sejumlah penempatan dan operasi pertukaran. Blok terakhir yang digunakan adalah 22-Mbyte blok tempat partisi 14-Mbyte dibuat. Gambar 4.5b menunjukkan perbedaan antara algoritma penempatan terbaik, pertama, dan berikutnya dalam memuaskan permintaan alokasi 16-Mbyte. Paling cocok akan mencari seluruh daftar blok yang tersedia dan manfaatkan blok 18-Mbyte, sisakan fragmen 2-Mbyte. Hasil pertandingan pertama dalam fragmen 6-Mbyte, dan hasil berikutnya-fit dalam fragmen 20-Mbyte.

Manakah dari pendekatan berikut yang terbaik akan bergantung pada urutan yang tepat dari pertukaran proses yang terjadi dan ukuran proses tersebut. Algoritma first-fit tidak hanya yang paling sederhana tetapi biasanya yang terbaik dan tercepat juga. Algoritma next-fit cenderung menghasilkan hasil yang sedikit lebih buruk daripada first-fit. Itu algoritma next-fit akan lebih sering mengarah pada alokasi dari blok bebas diakhir memori. Hasilnya adalah blok memori bebas terbesar, yang biasanya muncul di akhir ruang memori, dengan cepat dipecah menjadi fragmen kecil. Karenanya, pemadatan mungkin diperlukan lebih sering untuk Next-Fit. Di sisi lain, algoritma first-fit dapat mengotori bagian depan dengan partisi kecil yang bebas perlu dicari pada setiap umpan first-fit berikutnya. Algoritme best-fit, berkinerja terburuk. Karena algoritma ini mencari blok terkecil yang akan memenuhi persyaratan, yang menjamin fragmen tertinggal sekecil mungkin. Meskipun setiap permintaan memori selalu menyisakan jumlah memori terkecil, hasilnya adalah memori utama dengan cepat dikotori dengan blok yang terlalu kecil untuk memenuhi permintaan alokasi memori. Dengan demikian, pemadatan memori harus dilakukan lebih sering dibandingkan dengan algoritme lainnya.

Saat ini, pendekatan ini telah digantikan dengan teknik manajemen memori yang lebih canggih. Operasi penting Sistem yang menggunakan teknik ini adalah sistem operasi mainframe IBM, OS / MVT (Multiprogramming dengan Variabel Jumlah Tugas).



Gambar 4.5.

PENGGANTIAN ALGORITMA (Replacement Algoritme) Dalam sistem multiprogramming menggunakan dinamik partisi, akan tiba saatnya semua proses di memori utama dalam keadaan diblokir dan ada memori yang tidak cukup, bahkan setelah pemadatan. Untuk menghindari pemborosan waktu menunggu prosesor yang aktif proses untuk menjadi tidak diblokir, OS akan menukar salah satu proses keluar dari utama memori untuk memberikan ruang bagi proses baru atau untuk proses dalam status Siap-Tangguhkan. Oleh karena itu, sistem operasi harus memilih proses mana yang akan diganti.

4.1.3. Buddy Sistem

Baik skema partisi tetap dan dinamis memiliki kekurangan. Skema partisi tetap membatasi jumlah proses aktif dan mungkin menggunakan ruang secara tidak efisien jika ada kecocokan yang buruk antara ukuran partisi yang tersedia dan ukuran proses. Sebuah skema partisi dinamis lebih kompleks untuk dipelihara dan mencakup overhead pemadatan. Kompromi yang menarik adalah sistem buddy,

Dalam sistem buddy, blok memori tersedia dengan ukuran 2^K words, $L \leq K \leq U$, dimana

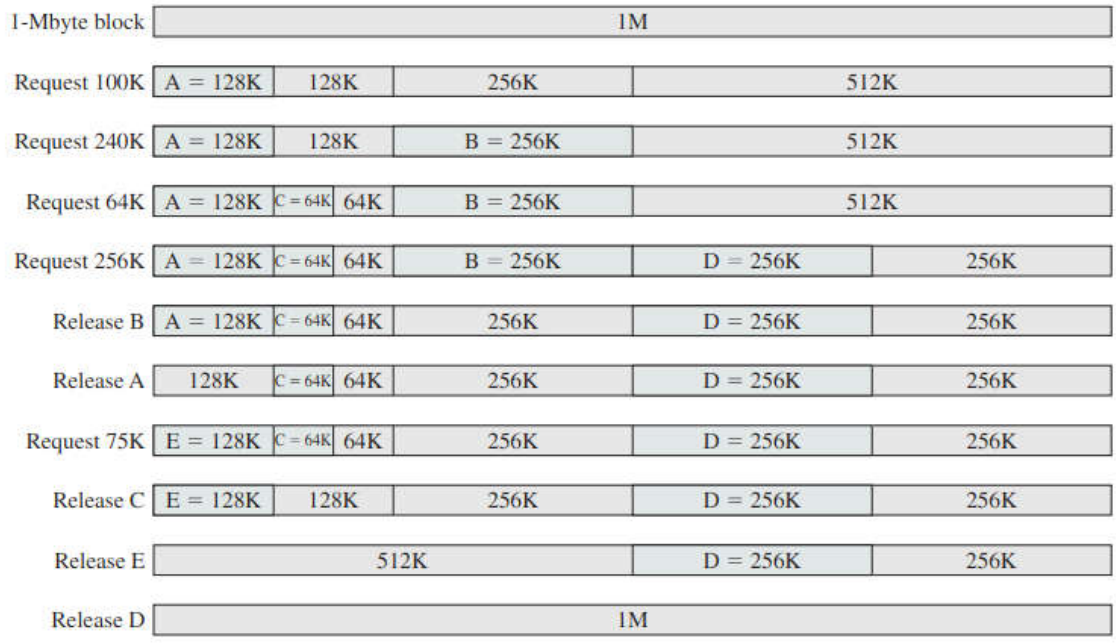
2^L blok ukuran terkecil yang dialokasikan

2^U blok ukuran terbesar yang dialokasikan; umumnya 2^U adalah ukuran keseluruhan memori tersedia untuk alokasi

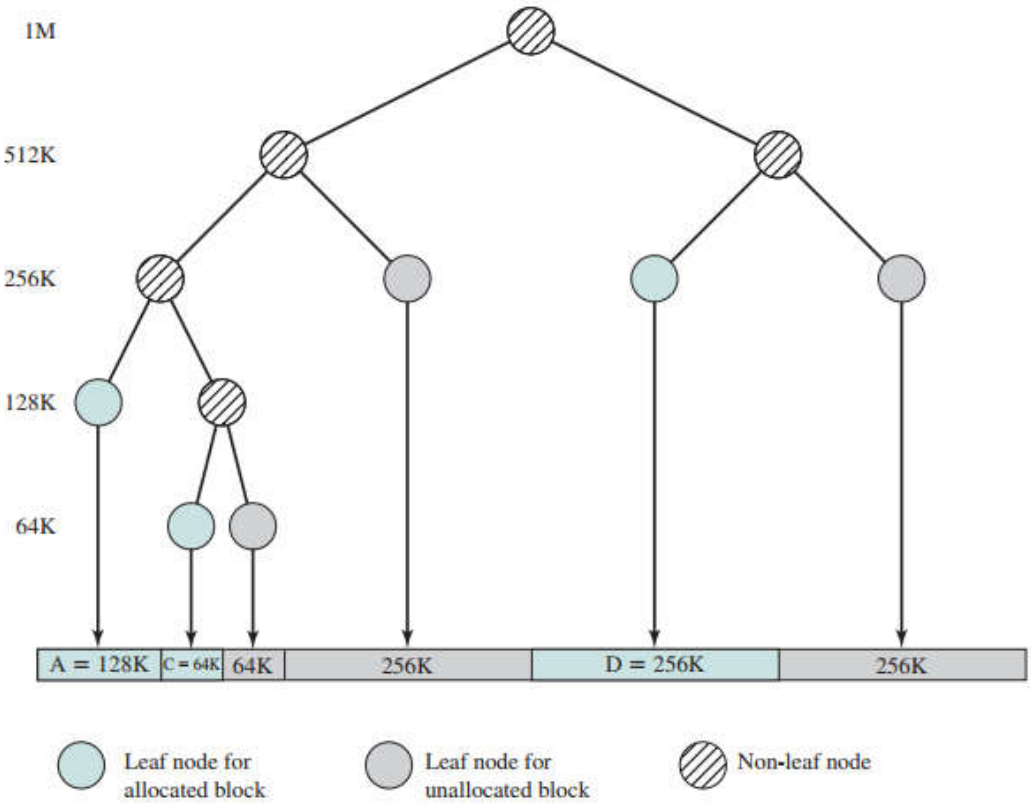
Untuk memulai, seluruh ruang yang tersedia untuk alokasi diperlakukan sebagai satu blok dari ukuran 2^U. Jika permintaan ukuran s sehingga $2^{U-1} < s \leq 2^U$ dibuat, maka seluruh blok dialokasikan. Jika tidak, blok akan dibagi menjadi dua *buddys* yang berukuran sama 2^{U-1}. Jika $2^{U-2} < s \leq 2^{U-1}$, maka permintaan dialokasikan ke salah satu dari dua buddys. Jika tidak, satu dari buddys dibagi menjadi dua lagi. Proses ini berlanjut hingga blok terkecil lebih besar dari atau sama dengan s dihasilkan dan dialokasikan ke permintaan. Kapan saja, file sistem buddy menyimpan daftar lubang (blok yang tidak terisi) untuk setiap ukuran 2^i. Lubang dapat dihapus dari daftar (i + 1) dengan membaginya menjadi dua untuk membuat dua buddys ukuran 2^i dalam daftar i. Setiap kali sepasang buddys di daftar menjadi tidak terisi, mereka dihapus dari daftar itu dan digabungkan menjadi satu blok di daftar (i + 1).

Gambar 4.6 menggambarkan contoh penggunaan blok awal 1-Mbyte. Permintaan pertama, A, adalah untuk 100 Kbytes, yang membutuhkan blok 128K. Blok awal dibagi menjadi dua buddys 512K. Yang pertama dibagi menjadi dua 256K buddys, dan yang pertama dibagi menjadi dua 128K buddys, salah satunya dialokasikan untuk A. Berikutnya request, B, membutuhkan blok 256K. Blok seperti itu sudah tersedia dan dialokasikan. Proses berlanjut dengan pemisahan dan penggabungan yang terjadi sesuai kebutuhan. Catatan, ketika E dilepaskan, dua 128K buddys digabungkan menjadi blok 256K.

Gambar 4.7 menunjukkan representasi pohon biner dari alokasi buddys segera setelah permintaan Rilis B. Simpul daun mewakili partisi saat ini dari memori. Jika dua buddys adalah simpul daun, maka setidaknya satu harus dialokasikan; jika tidak, mereka akan digabungkan menjadi blok yang lebih besar.



Gambar 4.6



Gambar 4.7.

Keuntungan:

Easy to find a partition for a program

Kerugian:

Internal fragmentation increases

C. Daftar Pustaka

1. Operating Systems Internals And Design Principles, Seventh Edition, Ch 7, William Stalling, Prentice Hall, 2012