Bab 4

PENJADWALAN PROSES

4.1. Pengertian dan Sasaran Penjadwalan Proses

Penjadwalan proses merupakan kumpulan kebijaksanaan dan mekanisme di sistem operasi yang berkaitan dengan urutan kerja yang dilakukan sistem komputer.

Adapun penjadwalan bertugas memutuskan :

a. Proses yang harus berjalan

b. Kapan dan selama berapa lama proses itu berjalan

Kriteria untuk mengukur dan optimasi kinerje penjadwalan :

a. Adil (fairness)

   Adalah proses-proses yang diperlakukan sama, yaitu mendapat jatah waktu pemroses yang sama dan tak ada proses yang tak kebagian layanan pemroses sehingga mengalami kekurangan waktu.

b. Efisiensi (eficiency)

   Efisiensi atau utilisasi pemroses dihitung dengan perbandingan (rasio) waktu sibuk pemroses.

c. Waktu tanggap (response time)

   Waktu tanggap berbeda untuk :

   c.1 Sistem interaktif

      Didefinisikan sebagai waktu yang dihabiskan dari saat karakter terakhir dari perintah dimasukkan atau transaksi sampai hasil pertama muncul di layar. Waktu tanggap ini disebut terminal response time.

   c.2 Sistem waktu nyata

      Didefinisikan sebagai waktu dari saat kejadian (internal atau eksternal) sampai instruksi pertama rutin layanan yang dimaksud dieksekusi, disebut event response time.

d. Turn around time

Adalah waktu yang dihabiskan dari saat program atau job mulai masuk ke sistem sampai proses diselesaikan sistem. Waktu yang dimaksud adalah waktu yang dihabiskan di dalam sistem, diekspresikan sebagai penjumlah waktu eksekusi (waktu pelayanan job) dan waktu menunggu, yaitu : Turn arround time = waktu eksekusi + waktu menunggu.

e. Throughput

Adalah jumlah kerja yang dapat diselesaikan dalam satu unit waktu. Cara untuk mengekspresikan throughput adalah dengan jumlah job pemakai yang dapat dieksekusi dalam satu unit/interval waktu.

Kriteria-kriteria tersebut saling bergantung dan dapat pula saling bertentangan sehingga tidak dimungkinkan optimasi semua kriteria secara simultan.

Contoh : untuk memberi waktu tanggap kecil memerlukan penjadwalan yang sering beralih ke antara proses-proses itu. Cara ini meningkatkan overhead sistem dan mengurangi throughput.

Oleh karena itu dalam menentukan kebijaksanaan perancangan penjadwalan sebaiknya melibatkan kompromi diantara kebutuhan-kebutuhan yang saling bertentangan. Kompromi ini bergantung sifat dan penggunaan sistem komputer.

Sasaran penjadwalan berdasarkan kriteria-kriteria optimasi tersebut :

a. Menjamin tiap proses mendapat pelayanan dari pemroses yang adil.
b. Menjaga agar pemroses tetap dalam keadaan sibuk sehingga efisiensi mencapai maksimum. Pengertian sibuk adalah pemroses tidak menganggur, termasuk waktu yang dihabiskan untuk mengeksekusi program pemakai dan sistem operasi.
c. Meminimalkan waktu tanggap.
d. Meminimalkan turn arround time.
e. Memaksimalkan jumlah job yang diproses persatu interval waktu. Lebih besar angka throughput, lebih banyak kerja yang dilakukan sistem.

4.2 Tipe Penjadwalan

Terdapat 3 tipe penjadwal berada secara bersama-sama pada sistem operasi yang kompleks, yaitu:

1. Penjadwal jangka pendek (short term scheduller)

Bertugas menjadwalkan alokasi pemroses di antara proses-proses ready di memori utama. Penjadwalan dijalankan setiap terjadi pengalihan proses untuk memilih proses berikutnya yang harus dijalankan.

2. Penjadwal jangka menengah (medium term scheduller)

   Setelah eksekusi selama suatu waktu, proses mungkin menunda sebuah eksekusi karena membuat permintaan layanan masukan/keluaran atau memanggil suatu system call. Proses-proses tertunda tidak dapat membuat suatu kemajuan menuju selesai sampai kondisi-kondisi yang menyebabkan tertunda dihilangkan. Agar ruang memori dapat bermanfaat, maka proses dipindah dari memori utama ke memori sekunder agar tersedia ruang untuk proses-proses lain. Kapasitas memori utama terbatas untuk sejumlah proses aktif.

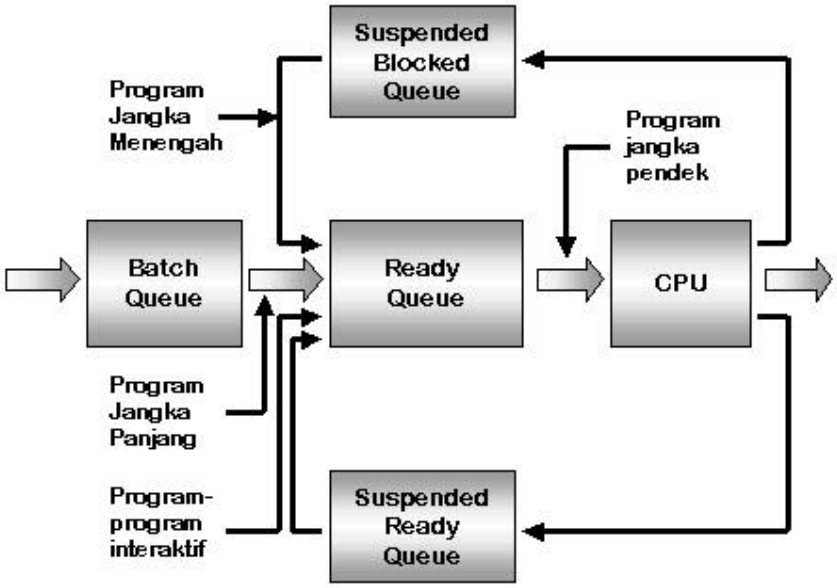
   Aktivitas pemindahan proses yang tertunda dari memori utama ke memori sekunder disebut swapping. Proses-proses mempunyai kepentingan kecil saat itu sebagai proses yang tertunda. Tetapi, begitu kondisi yang membuatnya tertunda hilang dan proses dimasukkan kembali ke memori utama dan ready.

3. Penjadwal jangka panjang (long term scheduller)

   Penjadwal ini bekerja terhadap antrian batch dan memilih batch berikutnya yang harus dieksekusi. Batch biasanya adalah proses-proses dengan penggunaan sumber daya yang intensif (yaitu waktu pemroses, memori, perangkat masukan/keluaran), program-program ini berprioritas rendah, digunakan sebagai pengisi (agar pemroses sibuk) selama periode aktivitas job-job interaktif rendah.

Sasaran penjadwalan berdasarkan tipe-tipe penjadwalan :

a. Memaksimumkan kinerja untuk memenuhi satu kumpulan kriteria yang diharapkan.
b. Mengendalikan transisi dari suspended to ready (keadaan suspend ke ready) dari proses-proses swapping.
c. Memberi keseimbangan job-job campuran.



Gambar 4.1 : Tipe-tipe penjadwalan

4.3 Strategi penjadwalan

Terdapat dua strategi penjadwalan, yaitu :

1. Penjadwalan nonpreemptive (run to completion)

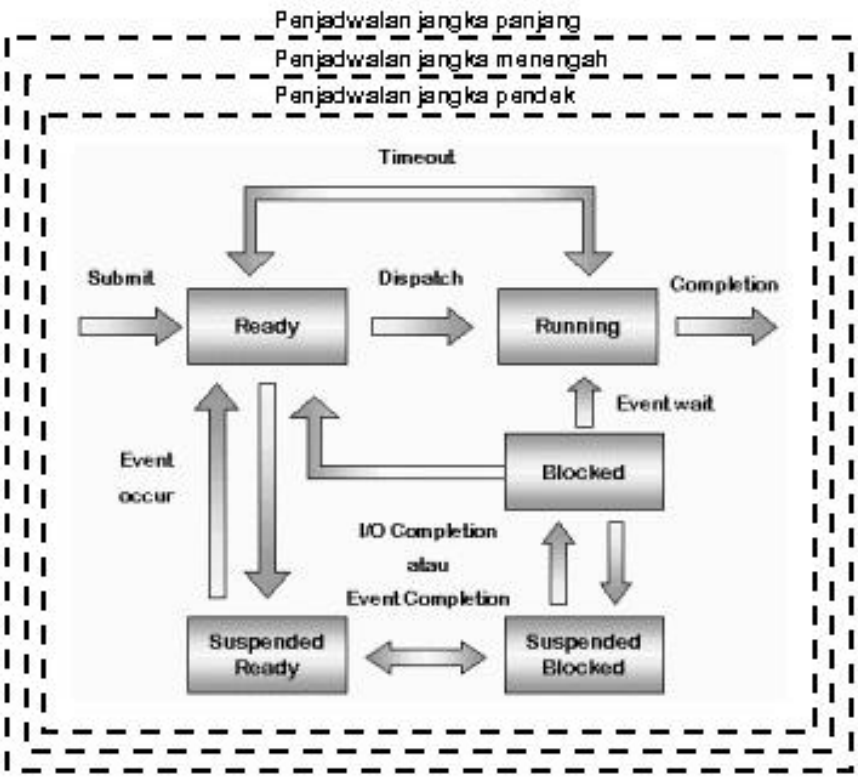
   Proses diberi jatah waktu oleh pemroses, maka pemroses tidak dapat diambil alih oleh proses lain sampai proses itu selesai.

2. Penjadwalan preemptive

   Proses diberi jatah waktu oleh pemroses, maka pemroses dapat diambil alih proses lain, sehingga proses disela sebelum selesai dan harus dilanjutkan menunggu jatah waktu pemroses tiba kembali pada proses itu. Berguna pada sistem dimana proses-proses yang mendapat perhatian/tanggapan pemroses secara cepat, misalnya :

   a. Pada sistem realtime, kehilangan interupsi (tidak layani segera) dapat berakibat fatal.
   b. Pada sistem interaktif, agar dapat menjamin waktu tanggap yang memadai. Penjadwalan secara preemptive baik tetapi harus dibayar mahal. Peralihan proses memerlukan overhead (banyak tabel yang dikelola). Supaya efektif, banyak proses harus berada di memori utama sehingga proses-proses tersebut dapat segera running begitu diperlukan. Menyimpan banyak proses

tak running benar-benar di memori utama merupakan suatu overhead tersendiri.



Gambar 4.2 : Tipe-tipe penjadwalan dikaitkan dengan diagram state

4.4 Algoritma-algoritma Penjadwalan

Berikut jenis-jenis algoritma berdasarkan penjadwalan :

1. Nonpreemptive, menggunakan konsep :

a. FIFO (First In First Out) atau FCFS (First Come First Serve)
b. SJF (Shortest Job First)
c. HRN (Highest Ratio Next)
d. MFQ (Multiple Feedback Queues)

2. Preemptive, menggunakan konsep :

a. RR (Round Robin)

b. SRF (Shortest Remaining First)

c. PS (Priority Schedulling)

d. GS (Guaranteed Schedulling)

Klasifikasi lain selain berdasarkan dapat/tidaknya suatu proses diambil secara paksa adalah klasifikasi berdasarkan adanya prioritas di proses-proses, yaitu :

1. Algoritma penjadwalan tanpa berprioritas.
2. Algoritma penjadwalan berprioritas, terdiri dari :
   a. Berprioritas statik
   b. Berprioritas dinamis

4.5 Algoritma Preemptive

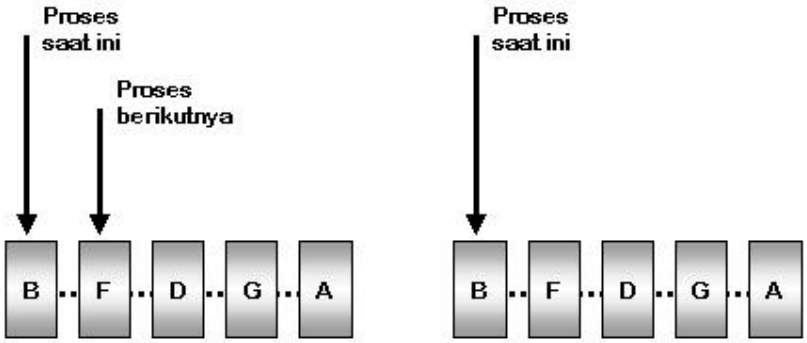
A. Round Robin (RR)

Merupakan :

- ☑ Penjadwalan yang paling tua, sederhana, adil,banyak digunakan algoritmanya dan mudah diimplementasikan.
- ☑ Penjadwalan ini bukan dipreempt oleh proses lain tetapi oleh penjadwal berdasarkan lama waktu berjalannya proses (preempt by time).
- ☑ Penjadwalan tanpa prioritas.
- ☑ Berasumsi bahwa semua proses memiliki kepentingan yang sama, sehingga tidak ada prioritas tertentu.

Semua proses dianggap penting sehingga diberi sejumlah waktu oleh pemroses yang disebut kwanta (quantum) atau time slice dimana proses itu berjalan. Jika proses masih running sampai akhir quantum, maka CPU akan mempreempt proses itu dan memberikannya ke proses lain.

Penjadwal membutuhkannya dengan memelihara daftar proses dari runnable. Ketika quantum habis untuk satu proses tertentu, maka proses tersebut akan diletakkan diakhir daftar (list), seperti nampak dalam gambar berikut ini :



Gambar 4.3

(a) : Daftar proses runnable.

(b) : Daftar proses runnable sesudah proses b habis quantumnya.

Algoritma yang digunakan :

1. Jika kwanta habis dan proses belum selesai, maka proses menjadi runnable dan pemroses dialihkan ke proses lain.
2. Jika kwanta belum habis dan proses menunggu suatu kejadian (selesainya operasi I/O), maka proses menjadi blocked dan pemroses dialihkan ke proses lain.
3. Jika kwanta belum habis tetapi proses telah selesai, maka proses diakhiri dan pemroses dialihkan ke proses lain.

Diimplementasikan dengan :

1. Mengelola senarai proses ready (runnable) sesuai urutan kedatangan.
2. Ambil proses yang berada di ujung depan antrian menjadi running.
3. Bila kwanta belum habis dan proses selesai, maka ambil proses di ujung depan antrian proses ready.
4. Jika kwanta habis dan proses belum selesai, maka tempatkan proses running ke ekor antrian proses ready dan ambil proses di ujung depan antrian proses ready.

Masalah yang timbul adalah menentukan besar kwanta, yaitu :

☑ Kwanta terlalu besar menyebabkan waktu tanggap besar dan turn arround time rendah.

☑ Kwanta terlalu kecil menyebabkan peralihan proses terlalu banyak sehingga menurunkan efisiensi proses.

Switching dari satu proses ke proses lain membutuhkan kepastian waktu yang digunakan untuk administrasi, menyimpan, memanggil nilai-nilai register, pemetaan memori, memperbaiki tabel proses dan senarai dan sebagainya. Mungkin proses switch ini atau konteks switch membutuhkan waktu 5 msec disamping waktu pemroses yang dibutuhkan untuk menjalankan proses tertentu.
Dengan permasalahan tersebut tentunya harus ditetapkan kwanta waktu yang optimal berdasarkan kebutuhan sistem dari hasil percobaan atau data historis. Besar kwanta waktu beragam bergantung beban sistem. Apabila nilai quantum terlalu singkat akan menyebabkan terlalu banyak switch antar proses dan efisiensi CPU akan buruk, sebaliknya bila nilai quantum terlalu lama akan menyebabkan respon CPU akan lambat sehingga proses yang singkat akan menunggu lama. Sebuah quantum sebesar 100 msec merupakan nilai yang dapat diterima.

Penilaian penjadwalan ini berdasarkan kriteria optimasi :

· Adil

Adil bila dipandang dari persamaan pelayanan oleh pemroses.

· Efisiensi

Cenderung efisien pada sistem interaktif.

· Waktu tanggap

Memuaskan untuk sistem interaktif, tidak memadai untuk sistem waktu nyata.

· Turn around time

Cukup baik.

· Throughtput

Cukup baik.

Penjadwalan ini :

a. Baik untuk sistem interactive-time sharing dimana kebanyakan waktu dipergunakan menunggu kejadian eksternal.
   Contoh : text editor, kebanyakan waktu program adalah untuk menunggu keyboard, sehingga dapat dijalankan proses-proses lain.
b. Tidak cocok untuk sistem waktu nyata apalagi hard-real-time applications.

B. Priority Schedulling (PS)

Adalah tiap proses diberi prioritas dan proses yang berprioritas tertinggi mendapat jatah waktu lebih dulu (running). Berasumsi bahwa masing-masing proses memiliki prioritas tertentu, sehingga akan dilaksanakan berdasar prioritas yang dimilikinya. Ilustrasi yang dapat memperjelas prioritas tersebut adalah dalam komputer militer, dimana proses dari jendral berprioritas 100, proses dari kolonel 90, mayor berprioritas 80, kapten berprioritas 70, letnan berprioritas 60 dan seterusnya. Dalam UNIX perintah untuk mengubah prioritas menggunakan perintah nice.

Pemberian prioritas diberikan secara :

a. Statis (static priorities)

Berarti prioritas tidak berubah.

Keunggulan :

- ☑ Mudah diimplementasikan.
- ☑ Mempunyai overhead relatif kecil.

Kelemahan :

- ☑ Tidak tanggap terhadap perubahan lingkungan yang mungkin menghendaki
- ☑ penyesuaian prioritas.

b. Dinamis (dynamic priorities)

Merupakan mekanisme untuk menanggapi perubahan lingkungan sistem beroperasi. Prioritas awal yang diberikan ke proses mungkin hanya berumur pendek setelah disesuaikan ke nilai yang lebih tepat sesuai lingkungan.

Kelemahan :

- Implementasi mekanisme prioritas dinamis lebih kompleks dan mempunyai overhead lebih besar. Overhead in diimbangi dengan peningkatan daya tanggap sistem.

Contoh penjadwalan berprioritas :

Proses-proses yang sangat banyak operasi masukan/keluaran menghabiskan kebanyakan waktu menunggu selesainya operasinya masukan/keluaran. Proses-proses ini diberi prioritas sangat tinggi sehingga begitu proses memerlukan pemroses segera diberikan, proses akan segera memulai permintaan masukan/keluaran berikutnya sehingga menyebabkan proses blocked menunggu selesainya operasi masukan/keluaran. Dengan demikian pemroses dapat

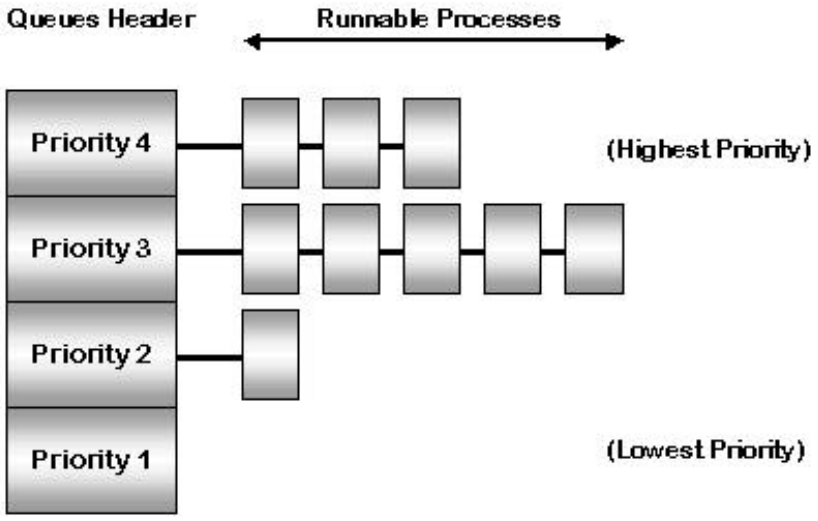
dipergunakan proses-proses lain. Proses-proses I/O berjalan paralel bersama proses-proses lain yang benar-benar memerlukan pemroses, sementara proses-proses I/O itu menunggu selesainya operasi DMA.
Proses-proses yang sangat banyak operasi I/O-nya, kalau harus menunggu lama untuk memakai pemroses (karena prioritas rendah) hanya akan membebani memori, karena harus disimpan tanpa perlu proses-proses itu dimemori karena tidak selesai-selesai menunggu operasi masukan dan menunggu jatah pemroses.

Dalam algoritma berprioritas dinamis dituntun oleh keputusan untuk memenuhi kebijaksanaan tertentu yang menjadi tujuan. Layanan yang bagus adalah menset prioritas dengan nilai 1/f, dimana f adalah ration kwanta terakhir yang digunakan proses.
Contoh :

· Proses yang menggunakan 2 msec kwanta 100 ms, maka prioritasnya50.
· Proses yang berjalan selama 50 ms sebelum blocked berprioritas 2.
· Proses yang menggunakan seluruh kwanta berprioritas 1.

Kebijaksanaan yang diterapkan adalah jaminan proses-proses mendapat layanan adil dari pemroses dalam arti jumlah waktu pemroses yang sama. Keunggulannya penjadwalan berpriorita adalah memenuhi kebijaksanaan yang ingin mencapai maksimasi suatu kriteria diterapkan. Algoritma ini dapat dikombinasikan, yaitu dengan mengelompokkan proses-proses menjadi kelas-kelas prioritas. Penjadwalan berprioritas diterapkan antar kelas-kelas proses itu.

Algoritma penjadwal akan menjalankan : proses runnable untuk prioritas 4 lebih dulu secara round robin, apabila kelas 4 semua sudah diproses, selanjutnya akan menjalankan proses runnable untuk prioritas 3 secara round robin, apabila kelas 3 semua sudah diproses (habis), selanjutnya akan menjalankan proses runnable untuk prioritas 2 secara round robin, dan seterusnya, seperti dalam gambar berikut :



Gambar 4.4 : Skedul algoritma denga empat klas prioritas

C. Multiple Feedback Queues (MFQ)

Merupakan :

☑ Penjadwalan berprioritas dinamis

Penjadwalan ini untuk mencegah (mengurangi) banyaknya swapping dengan proses-proses yang sangat banyak menggunakan pemroses (karena menyelesaikan tugasnya memakan waktu lama) diberi jatah waktu (jumlah kwanta) lebih banyak dalam satu waktu. Penjadwalan ini juga menghendaki kelas-kelas prioritas bagi proses-proses yang ada. Kelas tertinggi berjalan selama satu kwanta, kelas berikutnya berjalan selama dua kwanta, kelas berikutnya berjalan empat kwanta, dan seterusnya.

Ketentuan yang berlaku adalah sebagai berikut

☑ Jalankan proses pada kelas tertinggi.

☑ Jika proses menggunakan seluruh kwanta yang dialokasikan, maka diturunkan kelas prioritasnya.

☑ Proses yang masuk untuk pertama kali ke sistem langsung diberi kelas tertinggi.

Mekanisme ini mencegah proses yang perlu berjalan lama swapping berkali-kali dan mencegah proses-proses interaktif yang singkat harus menunggu lama.

D. Shortest Remaining First (SRF)

Merupakan :

- ☑ Penjadwalan berprioritas.dinamis.
- ☑ Adalah preemptive untuk timesharing
- ☑ Melengkapi SJF

  Pada SRF, proses dengan sisa waktu jalan diestimasi terendah dijalankan, termasuk proses-proses yang baru tiba.
- ☑ Pada SJF, begitu proses dieksekusi, proses dijalankan sampai selesai.
- ☑ Pada SRF, proses yang sedang berjalan (running) dapat diambil alih proses baru dengan sisa waktu jalan yang diestimasi lebih rendah.

Kelemahan :

- ☑ Mempunyai overhead lebih besar dibanding SJF. SRF perlu penyimpanan waktu layanan yang telah dihabiskan job dan kadang-kadang harus menangani peralihan.
- ☑ Tibanya proses-proses kecil akan segera dijalankan.
- ☑ Job-job lebih lama berarti dengan lama dan variasi waktu tunggu lebih lama dibanding pada SJF.

SRF perlu menyimpan waktu layanan yang telah dihabiskan , menambah overhead. Secara teoritis, SRF memberi waktu tunggu minimum tetapi karena overhead peralihan, maka pada situasi tertentu SFJ bisa memberi kinerja lebih baik dibanding SRF.

E. Guaranteed Scheduloing (GS)

Penjadwalan ini memberikan janji yang realistis (memberi daya pemroses yang sama) untuk membuat dan menyesuaikan performance adalah jika ada N pemakai, sehingga setiap proses (pemakai) akan mendapatkan 1/N dari daya pemroses CPU. Untuk mewujudkannya, sistem harus selalu menyimpan informasi tentang jumlah waktu CPU untuk semua proses sejak login dan juga berapa lama pemakai sedang login. Kemudian jumlah waktu CPU, yaitu waktu mulai login dibagi dengan n, sehingga lebih mudah menghitung rasio waktu CPU. Karena jumlah waktu pemroses

tiap pemakai dapat diketahui, maka dapat dihitung rasio antara waktu pemroses yang sesungguhnya harus diperoleh, yaitu 1/N waktu pemroses seluruhnya dan waktu pemroses yang telah diperuntukkan proses itu.

Rasio 0,5 berarti sebuah proses hanya punya 0,5 dari apa yang waktu CPU miliki dan rasio 2,0 berarti sebuah proses hanya punya 2,0 dari apa yang waktu CPU miliki. Algoritma akan menjalankan proses dengan rasio paling rendah hingga naik ketingkat lebih tinggi diatas pesaing terdekatnya. Ide sederhana ini dapat diimplementasikan ke sistem real-time dan memiliki penjadwalan berprioritas dinamis.

4.6 Algoritma Nonpreemptive

A. First In First Out (FIFO)

Merupakan :

· Penjadwalan tidak berprioritas.

FIFO adalah penjadwalan paling sederhana, yaitu :

· Proses-proses diberi jatah waktu pemroses berdasarkan waktu kedatangan.

· Pada saat proses mendapat jatah waktu pemroses, proses dijalankan sampai selesai.

Penilian penjadwalan ini berdasarkan kriteria optimasi :

· *Adil*

Adil dalam arti resmi (proses yang datang duluan akan dilayani lebih dulu), tapi dinyatakan tidak adil karena job-job yang perlu waktu lama membuat job-job pendek menunggu. Job-job yang tidak penting dapat membuat job-job penting menunggu lama.

· *Efisiensi*

Sangat efisien.

· *Waktu tanggap*

Sangat jelek, tidak cocok untuk sistem interaktif apalagi untuk sistem waktu nyata.

· *Turn around time*

Jelek.

· *Throughtput*

Jelek.

FIFO jarang digunakan secara mandiri, tetapi dikombinasikan dengan skema lain, misalnya : Keputusan berdasarkan prioritas proses. Untuk proses-pross berprioritas sama diputuskan berdasarkan FIFO.

Penjadwalan ini :

a. Baik untuk sistem batch yang sangat jarang berinteraksi dengan pemakai.
   Contoh : aplikasi analisis numerik, maupun pembuatan tabel.
b. Sangat tidak baik (tidak berguna) untuk sistem interaktif, karena tidak memberi waktu tanggap yang baik.
c. Tidak dapat digunakan untuk sistem waktu nyata (real-time applications).

B. Shortest Job First (SJF)

Penjadwalan ini mengasumsikan waktu jalan proses sampai selesai diketahui sebelumnya. Mekanismenya adalah menjadwalkan proses dengan waktu jalan terpendek lebih dulu sampai selesai, sehingga memberikan efisiensi yang tinggi dan turn around time rendah dan penjadwalannya tak berprioritas.

Contoh :

Terdapat empat proses (job) yaitu A,B,C,D dengan waktu jalannya masing-masing adalah 8,4,4 dan 4 menit. Apabila proses-proses tersebut dijalankan, maka turn around time untuk A adalah 8 menit, untuk B adalah 12, untuk C adalah 16 dan untuk D adalah 20. Untuk menghitung rata-rata turn around time seluruh proses adalah dengan menggunakan rumus :

$$(4a + 3b + 2c + 1d) / 4$$

Dengan menggunakan rumus, maka dapat dihitung turn around time-nya sebagai berikut (belum memperhatikan shortest job first, lihat gambar a) :

$$= (4a + 3b + 2c + 1d) / 4$$
$$= (4x8 + 3x4 + 2x4 + 1x4) / 4$$
$$= (32 + 12 + 8 + 4) / 4$$
$$= 56 / 4$$
$$= 14 \text{ menit}$$

Apabila keempat proses tersebut menggunakan penjadwalan shortest job fisrt (lihat gambar b), maka turn around time untuk B adalah 4, untuk C adalah 8, untuk D

adalah 12 dan untuk A adalah 20, sehingga rata-rata turn around timenya adalah sebagai berikut :

= (4a + 3b + 2c + 1d) / 4
= (4x4 + 3x4 + 2x4 + 1x8) / 4
= (16 + 12 + 8 + 8) / 4
= 44 / 4
= 11 menit

Tidak memperhatikan SJF

```
Posisi   : a     b   c   d
Priority : 4     3   2   1
Job      : A     B   C   D

        +-----------------+
        :  8  : 4 : 4 : 4 :
        +-----------------+
```

(a)

Memperhatikan SJF

```
  a   b   c    d
  4   3   2    1
  B   C   D    A

      +-----------------+
      : 4 : 4 : 4 :  8  :
      +-----------------+
```

(b)

Jelas bahwa a memberikan nilai kontribusi yang besar, kemudian b, c dan d. Karena SJF selalu memperhatikan rata-rata waktu respon terkecil, maka sangat baik untuk proses interaktif. Umumnya proses interaktif memiliki pola, yaitu menunggu perintah, menjalankan perintah, menunggu perintah dan menjalankan perintah, begitu seterusnya.

Masalah yang muncul adalah :

· Tidak mengetahui ukuran job saat job masuk.
  Untuk mengetahui ukuran job adalah dengan membuat estimasi berdasarkan kelakukan sebelumnya.
· Proses yang tidak datang bersamaan, sehingga penetapannya harus dinamis.
  Penjadwalan ini jarang digunakan, karena merupakan kajian teoritis untuk pembandingan turn around time.

C. Highest Ratio Next (HRN)

Merupakan :

- ☑ Penjadwalan berprioritas dinamis.
- ☑ Penjadwalan untuk mengoreksi kelemahan SJF.
- ☑ Adalah strategi penjadwalan dengan prioritas proses tidak hanya merupakan fungsi waktu layanan tetapi juga jumlah waktu tunggu proses. Begitu proses mendapat jatah pemroses, proses berjalan sampai selesai.

Prioritas dinamis HRN dihitung berdasarkan rumus :
Prioritas = (waktu tunggu + waktu layanan) / waktu layanan
Karena waktu layanan muncul sebagai pembagi, maka job lebih pendek berprioritas lebih baik, karena waktu tunggu sebagai pembilang maka proses yang telah menunggu lebih lama juga mempunyai kesempatan lebih bagus.
Disebut HRN, karena waktu tunggu ditambah waktu layanan adalah waktu tanggap, yang berarti waktu tanggap tertinggi yang harus dilayani.

4.7 Variasi yang diterapkan pada sistem waktu nyata (real time)

Karena sistem waktu nyata sering mempunyai deadline absolut, maka penjadwalan dapat berdasarkan deadline. Proses yang dijalankan adalah yang mempunyai deadline terdekat. Proses yang lebih dalam bahaya kehilangan deadline dijalankan lebih dahulu. Proses yang harus berakhir 10 detik lagi mendapat prioritas di atas proses yang harus berakhir 10 menit lagi.
Penjadwalan in disebut Earliest Deadline First (EDF).

4.8 Schedulling mechanism VS schedulling policy

Ada perbedaan antara schedulling mechanism dengan schedulling policy.
Skedul algoritma adalah dengan pemakaian nilai-nilai dalam parameter, dimana nilai-nilai parameter tersebut dapat diisi (set/change) oleh sebuah proses.
Kernel menggunakan algoritma schedulling priority dengan menyediakan sebuah system call dimana sebuah proses dapat diset dan diubah prioritasnya.

Metode ini dapat membantu proses induk (parent process) sehingga dapat mengontrol skedul anak prosesnya (child process). Disini mekanismenya adalah dalam kernel dan policy adalah penetapan nilai (set) oleh proses pemakai.

Bab 5

KONGKURENSI

5.1. Pengertian kongkurensi

Perkembangan sistem komputer mendatang adalah menuju ke sistem multi-processing, multiprogramming, terdistribusi dan paralel yang mengharuskan adanya proses-proses yang berjalan bersama dalam waktu yang bersamaan. Hal demikian merupakan masalah yang perlu perhatian dari perancang sistem operasi. Kondisi dimana pada saat yang bersamaan terdapat lebih dari satu proses disebut dengan kongkurensi (proses-proses yang kongkuren).

Proses-proses yang mengalami kongkuren dapat berdiri sendiri (independen) atau dapat saling berinteraksi, sehingga membutuhkan sinkronisasi atau koordinasi proses yang baik. Untuk penanganan kongkuren, bahasa pemograman saat ini telah memiliki mekanisme kongkurensi dimana dalam penerapannya perlu dukungan sistem operasi dimana bahasa berada.

5.2. Prinsip-prinsip kongkurensi

Kongkurensi merupakan kegiatan yang berhubungan dengan :

a. Alokasi waktu pemroses untuk proses-proses yang aktif.
b. Pemakaian bersama dan persaingan untuk mendapatkan sumber daya.
c. Komunikasi antar proses.
d. Sinkronisasi aktivitas banyak proses

Masalah kongkurensi dapat terjadi pada :

a. Banyak aplikasi.

Multiprogramming memungkinkan banyak proses sekaligus dijalankan.

Proses-proses dapat berasal dari aplikasi-aplikasi berbeda.

Pada sistem sistem multiprogramming bisa terdapat banyak aplikasi sekaligus yang dijalankan di sistem komputer.

b. Strukturisasi sebuah aplikasi yang terdiri dari kumpulan proses.

Perluasan prinsip perancangan modular dan pemograman terstruktur adalah suatu aplikasi dapat secara efektif diimplementasikan sebagai kumpulan proses. Dengan sekumpulan proses, maka tiap proses menyediakan satu layanan spesifik tertentu.

c. Strukturisasi sebuah proses.

Saat ini untuk peningkatan kinerja maka satu proses dapat memiliki banyak thread yang independen. Thread-thread tersebut harus dapat bekerjasama untuk mencapai tujuan proses.

Strukturisasi satu aplikasi dapat dilakukan dengan banyak proses atau banyak thread. Sistem operasi modern telah mendukung banyak thread yang berkinerja lebih bagus dibanding proses dalam kondisi/lingkungan yang lebih terkendali.

Contoh : Suatu word processor antara lain mempunyai kemampuan :

- ☑ Menerima masukan dari keyboard
- ☑ Menerima masukan dari mouse atau perangkat penunjuk yang lain (asinkron)
- ☑ Pemisahan kata-kata
- ☑ Memformat baris menjadi rata kanan, kiri atau kanan-kiri.

Aplikasi ini dapat diterapkan dengan banyak proses atau thread yang masing-masing mempunyai tugas tertentu. Dengan demikian, saat dilakukan penataan tampilan di layar, aplikasi sekaligus dapat menerima masukan dari mouse yang segera akan diteruskan ke aplikasi untuk mendapat perhatian.

d. Strukturisasi sistem operasi

Keunggulan strukturisasi dapat diterapkan ke pemrograman sistem.

Beberapa sistem operasi aktual yang dipasarkan dan yang sedang dalam riset telah diimplementasikan sebagai kumpulan proses. Sistem operasi bermodelkan client/server.

5.3. Kesulitan-kesulitan dalam kongkurensi

Kecepatan proses pada sistem dipengaruhi oleh :

a. Aktivitas-aktivitas proses-proses lain.
b. Cara sistem operasi menangani interupsi.
c. Kebijaksanaan penjadwalan yang dilakukan oleh sistem operasi

Beberapa kesulitan yang muncul :

a. Pemakaian bersama sumber daya global.

Apabila terdapat dua proses yang menggunakan variabel global yang sama serta keduanya membaca dan menulis ke variabel itu,maka urutan terjadinya pembacaan dan penulisan terhadap variabel itu menjadi kritis.

b. Pengelolaan alokasi sumber daya agar optimal.

Apabila proses A meminta suatu kanal masukan/keluaran tertentu dan dipenuhi, kemudian terjadi proses A di suspend sebelum menggunakan kanal tersebut. Jika sistem operasi mengunci kanal (tidak memperbolehkan atau mencegah proses lain untuk menggunakannya), maka tindakan tersebut menghasilkan inefisiensi.

c. Pencarian kesalahan pemrograman.

Pencarian kesalahan pada pemograman kongkuren lebih sulit dibanding pencarian kesalahan pada program-program sekuen.

Penanganan kongkurensi adalah dengan :

a. Mengetahui proses-proses yang aktif.

Sistem operasi mengelola senarai proses di sistem operasi. Senarai ini berupa senarai PCB proses. Senarai berjumlah sesuai jumlah state yang diimplementasikan sistem operasi.

b. Mengatur alokasi dan dealokasi beragam sumber daya untuk tiap proses yang aktif.

Sumber daya yang harus dikelola antara lain :

- ☑ Waktu pemroses
- ☑ Memori
- ☑ Berkas-berkas (file)
- ☑ Peralatan masukan/keluaran
- ☑ Dan sebagainya

c. Proteksi data dan sumber daya fisik proses.

Proteksi data dan sumber daya fisik masing-masing proses dari gangguan (interfensi) proses-proses lain.

d. Hasil-hasil proses harus independen.

Hasil-hasil proses harus independen terhadap kecepatan relatif proses-proses lain dimana eksekusi dilakukan.

5.4. Mutual Exclusion

Merupakan kondisi dimana terdapat sumber daya yang tidak dapat dipakai bersama pada waktu yang bersamaan (misalnya : printer, disk drive). Kondisi demikian disebut sumber daya kritis, dan bagian program yang menggunakan sumber daya kritis disebut critical region / section. Hanya satu program pada satu saat yang diijinkan masuk ke critical region. Pemogram tidak dapat bergantung pada sistem operasi untuk memahami dan memaksakan batasan ini, karena maksud program tidak dapat diketahu oleh sistem operasi.

Hanya saja, sistem operasi menyediakan layanan (system call) yang bertujuan untuk mencegah proses lain masuk ke critical section yang sedang digunakan proses tertentu. Pemograman harus menspesifikasikan bagian-bagian critical section, sehingga sistem operasi akan menjaganya.

Pemaksaan atau pelanggaran mutual exclusion menimbulkan :

a. Deadlock

b. Startvation

5.5. Deadlock

Ilustasi deadlock, misalnya :

- ☑ Terdapat dua proses, yaitu P1 dan P2 dan dua sumber daya kritis, yaitu R1 dan R2.
- ☑ Proses P1 dan P2 harus mengakses kedua sumber daya tersebut, dengan kondisi ini terjadi : R1 diberikan ke P1, sedangkan R2 diberikan ke P2.

Karena untuk melanjutkan eksekusi memerlukan kedua sumber daya sekaligus maka kedua proses akan saling menunggu sumber daya lain selamanya.

Tak ada proses yang dapat melepaskan sumber daya yang telah dipegangnya karena menunggu sumber daya lain yang tak pernah diperolehnya.

Kedua proses dalam kondisi deadlock, yang tidak dapat membuat kemajuan apapun dan deadlock merupakan kondisi terparah karena dapat melibatkan banyak proses dan semuanya tidak dapat mengakhiri prosesnya secara benar.

5.6. Startvation

Ilustasi deadlock, misalnya :

- ☑ Terdapat tiga proses, yaitu P1, P2 dan P3.
- ☑ P1, P2 dan P3 memerlukan pengaksesan sumber daya R secara periodik

Skenario berikut terjadi :

- ☑ P1 sedang diberi sumber daya R sedangkan P2 dan P3 diblocked menunggu sumber daya R.
- ☑ Ketika P1 keluar dari critical section, maka P2 dan P3 diijinkan mengakses R.
- ☑ Asumsi P3 diberi hak akses, kemudian setelah selesai, hak akses kembali diberikan ke P1 yang saat itu kembali membutuhkan sumber daya R.

Jika pemberian hak akses bergantian terus-menerus antara P1 dan P3, maka P2 tidak pernah memperoleh pengaksesan sumber daya R. Dalam kondisi ini memang tidak terjadi deadlock, hanya saja P2 mengalami starvation (tidak ada kesempatan untuk dilayani).

5.7. Interaksi antar proses

Pada sistem dengan banyak proses kongkuren, terdapat tiga kategori interaksi, yaitu :

a. Proses-proses saling tidak peduli (independen)

Proses-proses ini tidak dimaksudkan untuk bekerja bersama untuk mencapai tujuan tertentu. Pada multiprogramming dengan proses-proses independen, dapat berupa batch atau sesi interaktif, atau campuan keduanya.

Meski proses-proses tidak bekerja bersama, sistem operasi perlu mengatur persaingan diantara proses-proses itu dalam memperoleh sumber daya yang terbatas.

Contoh : Terdapat dua aplikasi yang berusaha mengakses printer yang sama, bila kedua aplikasi benar-benar mengakses printer yang sama secara bersamaan, maka kedua proses akan memperoleh hasil yang tak dikehendaki.

Sistem operasi harus mengatur pengaksesan-pengaksesan sumber daya agar tidak menyebabkan hasil yang tidak dikehendaki.

b. Proses-proses saling mempedulikan secara tidak langsung

Dimana proses-proses tidak perlu saling mempedulikan identitas proses-proses lain tapi sama-sama mengakses objek tertentu, seperti buffer masukan/keluaran. Proses-proses itu perlu bekerja sama (cooperation) dalam memakai bersama objek tertentu.

c. Proses-proses saling mempedulikan secara langsung.

Proses-proses dapat saling berkomunikasi dan dirancang bekerja sama untuk suatu aktivitas.

Interaksi antara proses-proses dan masalah-masalah yang harus diatasi dapat dilihat dalam tabel berikut :

Tabel 4.1 : Interaksi antara proses-proses dan permasalahan yang timbul

Derajat Kepedulian	**Hubungan**	**Akibat satu proses terhadap lainnya**	**Masalah pengendalian yang dilakukan**
Proses tak peduli	persaingan	• Hasil satu proses independen terhadap aksi proses lain. • Pewaktuan proses dapat berdampak pada proses lain	• Mutual exclusion. • Deadlock • Starvation
Proses secara tidak langsung peduli terhadap proses lain, yaitu obyek yang dipakai bersama	Kerjasama dengan pemakaian bersamaan	• Hasil-hasil proses dapat bergantung pada informasi yang diperoleh dari proses lain. • Pewaktuan proses dapat berdampak pada proses lain	• Mutual exclusion. • Deadlock • Starvation • Koherensi data
Proses secara langsung peduli terhadap proses	Kerjasama dengan komunikasi	• Hasil-hasil proses dapat bergantung pada informasi	• Deadlock • Starvation

lain (tersedia primitif-primitif untuk proses tersebut)		yang diperoleh dari proses lain. • Pewaktuan proses dapat berdampak pada proses lain	

5.8. Persaingan diantara proses-proses untuk sumber daya

Proses-proses kongkuren berkompetisi ketika proses-proses bersaing menggunakan sumber daya yang sama. Dua proses atau lebih perlu mengakses sumber daya yang sama pada suatu saat. Masing-masing proses tidak peduli keberadaan proses-proses lain dan masing-masing proses tidak dipengaruhi proses-proses lain.

Pada proses-proses berkompetisi ini, tidak ada pertukaran informasi antara proses-proses itu. Eksekusi satu proses dapat berpengaruh terhadap kelakuan proses-proses yang berkompetisi. Jika dua proses ingin mengakses satu sumber daya tunggal maka sistem operasi mengalokasikan untuk salah satu proses dan mengharuskan proses lain menunggu. Proses yang ditolak pengaksesan menjadi melambat.

Kasus ekstrim yang dapat terjadi adalah proses di-blocked terus-menerus sehingga tak pernah mengakses sumber daya. Proses tak pernah dapat berakhir dengan suskses. Kondisi tidak pernah dapat kesempatan dialokasikan sumber daya disebut startvation. Sistem operasi harus menghindarkan terjadinya kondisi ini.

Persaingan proses-proses untuk memperoleh sumber daya menimbulkan tiga masalah :

1. Mutual exclusion
2. Deadlock
3. Startvation

Pengendalian persaingan melibatkan sistem operasi, yang bertugas mengalokasikan sumber daya. Proses-proses itu sendiri harus menyatakan keperluan mutual exclusion (diprogram oleh pemrogram menggunakan system call

yang disediakan sistem operasi) dan sistem operasi menangani agar tidak terlanggar kondisi mutual exclusion, serta tidak terjadi deadlock dan startvation.

5.9. Kerjasama diantara proses-proses dengan pemakaian bersama

Dalam kasus kerjasama pemakaian sumber daya bersama meliputi proses-proses yang saling berinteraksi tanpa dinyatakan secara eksplisit.
Contoh : Banyak proses mengakses variabel atau berkas yang dipakai bersama.
Proses-proses dapat menggunakan dan memperbarui data yang dipakai bersama tanpa peduli proses-proses lain. Proses mengetahui bahwa proses-proses lain dapat juga mengakses data yang sama. Proses-proes harus bekerja sama untuk menjamin integritas data yang dipakai bersama tersebut.

Kerjasama diantara proses-proses dalam pemakaian bersama mempunyai masalah antara lain :

- Mutual exclusion
- Deadlock
- Startvation

Karena data disimpan pada suatu sumber daya (peralatan, memori), maka terdapat masalah pengendalian mutual exclusion, deadlock dan startvation.
Perbedaannya adalah item-item data dapat diakses dengan dua mode, yaitu :
1. Operasi pembacaan dan penulisan harus mutually exclusive (yaitu benar-benar hanya satu proses yang berada di critical section).
2. Operasi penulisan saja yang harus mutually exclusive.
   Pada situasi ini, masalah baru muncul yaitu mengenai koherensi data.
   Critical section digunakan untuk menjamin integritas data.

5.10. Kerjasama diantara proses-proses dengan komunikasi

Pada kasus persaingan, proses-proses memakai sumber daya tanpa peduli proses-proses lain. Pada kasus kedua, proses-proses memakai bersama nilai dan meski masing-masing proses tidak secara eksplisit peduli proses-proses lain. Tapi proses-proses peduli untuk menjaga integritas data.

Ketiak proses-proses bekerja sama dengan komunikasi, beragam proses berpartisipasi dalam suatu usaha dengan menghubungkan semua proses. Komunikasi menyediakan cara untuk sinkronisasi atau koordinasi beragam aktivitas. Komunikasi dicirikan dengan berisi pesan-pesan dengan suatu urutan. Primitif untuk mengirim dan menerima pesan disediakan sebagai bagian bahasa pemrograman atau disediakan kernel sistem operasi. Karena tak ada sesuatu yang dipakai bersama diantara proses-proses itu dalam melewatkan pesan-pesan, tak ada masalah mutual exclusion. Tetapi masalah deadlock dan startvation dapat muncul.

5.11. Pokok penyelesaian masalah kongkurensi

Pada dasarnya penyelesaian masalah kongkurensi terbagi menjadi dua, yaitu :
a. Mengasumsikan adanya memori yang digunakan bersama.
b. Tidak mengasumsikan adanya memori yang digunakan bersama.
Adanya memori bersama lebih mempermudah penyelesaian masalah kongkurensi. Metode penyelesaian ini dapat dipakai untuk sistem singleprocessor ataupun multiprocessor yang mempunyai memori bersama. Penyelesaian ini tidak dapat digunakan untuk multiprocessor tanpa memori bersama ataupun untuk sistem tersebar.

Bab 6

MANAJEMEN MEMORI

Bagian operating sistem yang mengatur memori disebut dengan memory manager. Pemakaian memori (manajemen memori dan organisasi) perlu dilakukan karena hal tersebut sangat mempengaruhi kinerja komputer, sehingga memiliki fungsi dan tugas penting dan kompleks yaitu berkaitan dengan :

a. Memori utama sebagai sumber daya yang harus dialokasikasikan dan dipakai bersama di antara sejumlah proses yang aktif, sehingga dapat memanfaatkan pemroses dan fasilitas masukan/keluaran secara efisien, sehingga memori dapat menampung sebanyak mungkin proses.
b. Upaya agar pemogram atau proses tidak dibatasi kapasitas memori fisik di sistem komputer.

6.1. Manajemen memori

Sistem manajemen memori dapat dibagi kedalam dua kelas, yaitu : pemindahan proses (back and forth) diantara memori utama dengan disk selama eksekusi (swapping and paging) dan tidak ada pemindahan proses.

Mempunyai beberapa fungsi, antara lain :

a. Mengelola informasi memori yang dipakai dan tidak dipakai.
b. Mengalokasikan memori ke proses yang memerlukan.
c. Mendealokasikan memori dari proses yang telah selesai.
d. Mengelola swapping antara memori utama dan disk.

6.2. Manajenen memori pada sistem multiprogramming

Untuk sistem komputer yang berukuran besar (bukan small computers), membutuhkan pengaturan memori, karena dalam multiprogramming akan melibatkan banyak pemakai secara simultan sehingga di memori akan terdapat lebih dari satu proses bersamaan. Oleh karena itu dibutuhkan sistem operasi yang mampu mendukung dua kebutuhan tersebut, meskipun hal tersebut saling bertentangan, yaitu :

a. Pemisahan ruang-ruang alamat.
b. Pemakaian bersama memori.

Manajer memori harus memaksakan isolasi ruang-ruang alamat tiap proses agar mencegah proses aktif atau proses yang ingin berlaku jahat mengakses dan merusak ruang alamat proses lain. Manajer memori di lingkungan multiprogramming sekalipun melakukan dua hal, yaitu :

a. Proteksi memori dengan isolasi ruang-ruang alamat secara disjoint.

c. Pemakaian bersama memori.

Memungkinkan proses-proses bekerja sama mengakses daerah memori bersama.

Ketika konsep multiprogramming digunakan, pemakaian CPU dapat ditingkatkan.

Sebuah model untuk mengamati pemakaian CPU secara probabilistic :

CPU utilization = 1 - p n

Dengan :

* N menunjukkan banyaknya proses pada suatu saat, sehingga kemungkinan bahwa semua n proses akan menunggu menggunakan I/O (masalah CPU menganggur) adalah sebesar p n. Fungsi dari n disebut sebagai degree of multiprogramming.
* P menunjukkan besarnya waktu yang digunakan sebuah proses

Contoh analisisnya :

Diketahui :

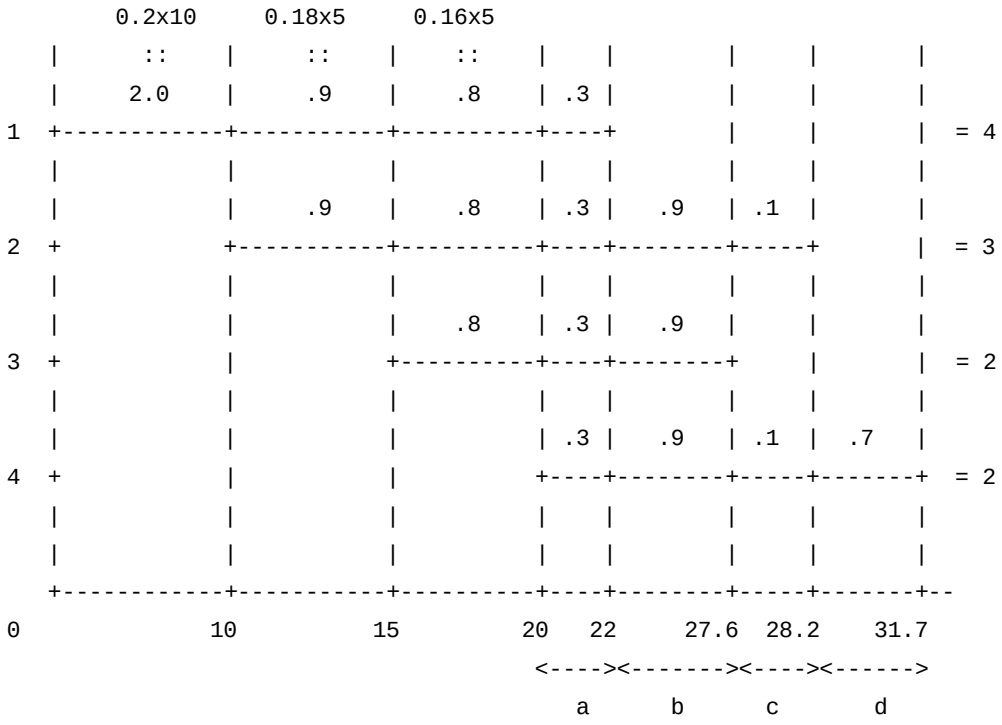
Job	Arrival time	CPU minutes needed
1	10:00	4
2	10:10	3
3	10:15	2
4	10:20	2

Bila semua job bersifat 80% I/O wait. Tentukan kapan waktu selesainya masing-masing job !

Jawab :

Process

```
                  1        2        3        4
-------------------------------------------------
CPU idle .80      .64      .51      .41
CPU busy .20      .36      .49      .59
CPU/process       .20      .18      .16      .15
-------------------------------------------------
```



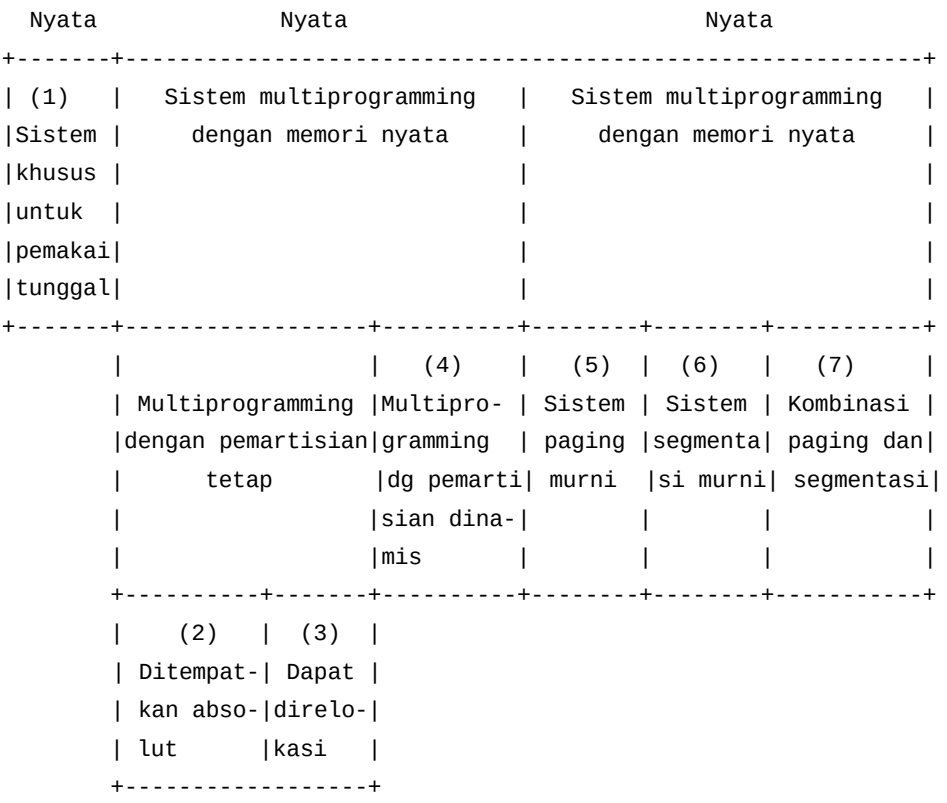
Keterangan :

```
ax0.15=0.3   bx0.16=0.9   cx0.18=0.1   dx0.2=0.7
     a=2          b=5.6        c=0.6       d=3.5
```

6.3. Klasifikasi manajemen memori.

Klasifikasi manajemen memori diberikan Deitel (DEI-90).

Gambar 6.1 menunjukkan skema klasifikasi manajamen memori.



Gambar 6.1 : Klasfikasi manajemen memori

Teknik-teknik manajemen memori (1), (2), (3), (4) merupakan pengelolaan untuk dengan kapasitas memori sebatas memori fisik yang tersedia.

Teknik-teknik ini tidak dapat digunakan untuk memuat program-program lebih besar dibanding kapasitas fisik memori yang tersedia.

Teknik-teknik manajemen memori (5), (6), (7) dapat digunakan untuk mengakali kapasitas memori yang terbatas sehingga dapat dijalankan program yang lebih besar dibanding kapasitas memori fisik yang tersedia.

6.4. Manajemen memori berdasarkan keberadaan swapping

Manajemen memori berdasarkan keberadaan swapping terbagi menjadi dua, yaitu :

1. Manajemen tanpa swapping.

Manajemen memori tanpa pemindahan citra proses antara memori utama dan disk selama ekseskusi.

2. Manajemen dengan swapping.

Manajemen memori dengan pemindahan citra proses antara memori utama dan disk selama ekseskusi.

6.5. Manajemen memori berdasar alokasi memori

Manajemen memori berdasar alokasi memori terbagi dua, yaitu :

1. Alokasi memori berurutan (kontigu).

Adalah tiap-tiap proses menempati satu blok tunggal lokasi memori yang berturutan.

Keunggulan :

a. Sederhana.
b. Tidak akan terbentuk lubang-lubang memori bersebaran.
c. Karena berurutan, proses dapat dieksekusi dengan cepat.

Kelemahan :

a. Dapat memboroskan memori.
b. Tidak dapat memuatkan proses bila tidak ada satu blok memori yang mencukupi.

2. Alokasi memori tak berurutan (non-kontinyu).

Program dibagi menjadi beberapa blok atau segmen. Blok-blok program ditempatkan di memori dalam potongan-potongan tanpa perlu saling berdekatan. Teknik biasa digunakan pada sistem memori maya sebagai alokasi page-page dilakukan secara global.

Keuntungan :

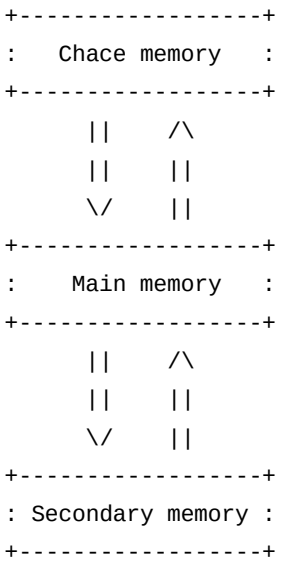
a. Sistem dapat memanfaatkan memori utama secara lebih efisien.
b. Sistem operasi masih mampu memuatkan proses bila jumlah total lubang-lubang memori cukup untuk memuat proses yang akan dieksekusi.

Kelemahan :

a. Memerlukan pengendalian yang lebih rumit dan sulit.
b. Memori dapat menjadi banyak lubang tersebar (memori tak terpakai bersebaran).

6.6. Hirarki memori

Pemakaian memori dua tingkat, menggunakan cache memory yang dapat meningkatkan kinerja dan utilisasi memori secara dinamik. Chace memory merupakan penyimpan berkecepatan tinggi lebih cepat dibanding memori utama.
Chace memory lebih mahal dibanding memori utama, sehingga kapasitas cache relatif kecil.



Gambar 6.2 : Hubungan chace memori, memori utama dan memori sekunder.

Gambar 6.2 memperlihatkan hubungan antara chace memory, memori utama dan penyimpan sekunder. Dengan cache memory, bagian program yang akan digunakan (dieksekusi atau diacu) dikopi dulu ke chace sebelum dieksekusi. Di chace memory, instruksi dapat dieksekusi dengan lebih cepat dibanding di memori utama. Penggunaan chace atau memori antara yang lebih cepat mempunyai alasan yang dikemukakan oleh Denning, yaitu ekseksui program biasanya pada suatu

interval waktu mengumpul di satu lokasi kecil. Prinsip ini disebut lokalitas. Lokalitas dapat berupa lokalitas waktu dan ruang. Prinsip lokalitas berkembang menjadi konsep working set model.

6.7. Manajemen memori tanpa swapping

Manajemen memori tanpa swapping terdiri dari :

a. Monoprogramming.

Monoprogramming sederhana tanpa swapping merupakan manajemen memori paling sederhana, sistem komputer hanya mengijinkan satu program/pemakai berjalan pada satu waktu. Semua sumber daya sepenuhnya dkuasi proses yang sedang berjalan.

Manajemen memori monoprogramming sederhana mempunyai ciri-ciri berikut :

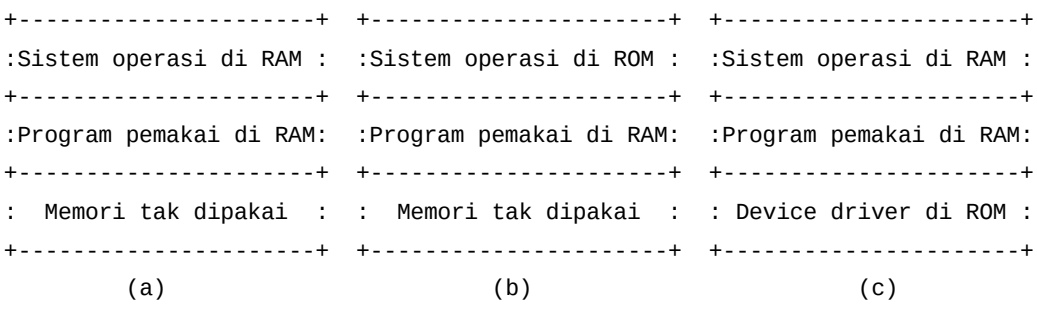
a. Hanya terdapat satu proses pada satu saat, sehingga proses baru akan menimpa proses lama yang sudah selesai eksekusi.
b. Hanya satu proses mengunakan semua memori.
c. Pemakai memusatkan program keseluruh memori dari disk atau tape.
d. Program mengambil kendali seluruh mesin.

Karena hanya terdapat satu proses dan menguasai seluruh sistem, maka eksekusi memori dilakukan secara berurutan.

Teknik ini digunakan sampai sekitar 1960, ditinggalkan bahkan untuk komputer pribadi karena tiap proses harus berisi device driver perangkat I/O yang digunakan.

Gambar 6.3 menunjukkan tiga organisasi memori menjalankan satu proses tunggal :

c. Gambar 6.3(a) menunjukkan seluruh kebutuhan (sistem operasi, device driver dan proses driver dapat ditempatkan di sistem operasi atau di setiap proses pemakai, bergantung perancang sistem operasi.
d. Gambar 6.3(b) menunjukkan sistem operasi ditempatkan di ROM, sedang program pemakai di RAM.
e. Gambar 6.3(c) menunjukkan device driver di ROM. Device driver di ROM biasa disebut ROM-BIOS (Read Only Memory - Basic Input Output Systems).



Gambar 6.3 : Tiga cara organisasi memori untuk satu proses tunggal

Embedded system

Teknik monoprogamming masih dipakai untuk sistem kecil yaitu sistem tempelan (embedded system) yang menempel atau terdapat disistem lain. Sistem-sistem tempelan menggunakan mikroprosesor kecil, seperti Intel 8051, dan sebagainya. Sistem ini biasanya untuk mengendalikan satu alat sehingga menjadi bersifat intelejen (intelegent devices) dalam menyediakan satu fungsi spesifik. Karena hanya satu fungsi spesifik, dapat diprogram di mikroprosesor dengan memori kecil (1-64 Kb).

Sistem tempelan telah banyak digunakan, misalnya sistem tempelan di mobil antar lain untuk :

a. Pengendalian pengapian.
b. Pengendalian pengeluaran bahan bakar.
c. Pengendalian pengereman.
d. Pengendalian suspensi.
e. Pengendalian kemudi.
f. Dan sebagainya.

Pada mobil mewah terdapat lebih dari 50 mikroprosesor, masing-masing mengendalikan satu fungsi spesifik.

Proteksi pada monoprogramming sederhana.

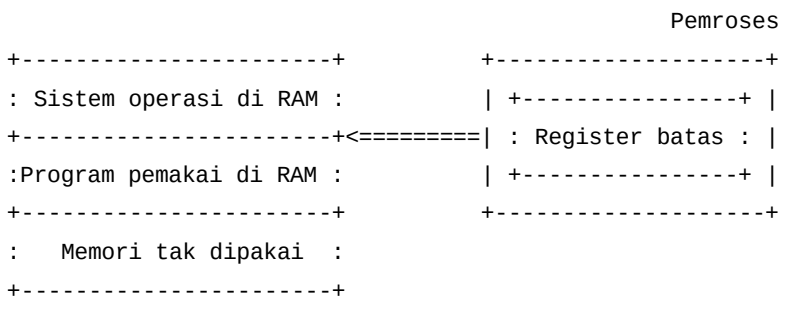
Pada monoprogramming, pemakai mempunyai kendali penuh terhadap seluruh memori utama. Memori terbagi menjadi tiga bagian, yaitu :

a. Bagian yang berisi rutin-rutin sistem operasi.

b. Bagian yang berisi program pemakai.
c. Bagian yang tidak digunakan.

Masalah proteksi di monoprogramming adalah cara memproteksi rutin sistem operasi dari penghancuran program pemakai. Program pemakai dapat tersesat sehingga memanipulasi atau menempati ruang memori rutin sistem operasi. Aktivitas program pemakai ini dapat merusak sistem operasi.
Sistem operasi harus diproteksi dari modifikasi program pemakai. Proteksi ini diimplementasikan menggunakan satu registe batas (boundary register) dipemroses.Setiap kali program pemakai mengacu alamat memori dibandingkan register batas untuk memastikan proses pemakai tidak merusak sistem operasi, yaitu tidak melewati nilai register batas.

Register batas berisi alamat memori tertinggi yang dipakai sistem operasi. Jika program pemakai mencoba memasuki sistem operasi, instruksi diintersepsi dan job diakhiri dan diberi pesan kesalahan. Untuk memperoleh layanan sistem operasi, program pemakai harus menggunakan instruksi spesifik meminta layanan sistem operasi. Integritas sistem operasi terjaga dan program pemakai tidak merusak bagian sistem operasi.



Gambar 6.4 : Proteksi pada monoprogramming

Gambar 6.4 menunjukkan skema proteksi menggunakan register batas.
Register batas menunjuk alamat terakhir sistem operasi. Bila program pemakai mengacu ke alamat daerah sistem operasi, pemroses menjadi fault menyatakan terjadinya pelanggaran pengaksesan oleh proses pemakai.

b. Multiprogramming dengan pemartisian statis.

Terdapat beberapa alasan kenapa multiprogramming digunakan, yaitu :

a. Mempermudah pemogram.
   Pemogram dapat memecah program menjadi dua proses atau lebih.
b. Agar dapat memberi layanan interaktif ke beberapa orang secara simultan.
   Untuk itu diperlukan kemampuan mempunyai lebih dari satu proses dimemori agar memperoleh kinerja yang baik.
c. Efisiensi penggunaan sumber daya.
   Bila pada multiprogramming maka proses tersebut diblocked (hanya DMA yang bekerja) dan proses lain mendapat jatah waktu pemroses, maka DMA dapat meningkatkan efisiensi sistem.
d. Eksekusi lebih murah jika proses besar dipecah menjadi beberapa proses kecil.
e. Dapat mengerjakan sejumlah job secara simultan.

Multiprogramming dapat dilakukan dengan pemartisian statis, yaitu memori dibagi menjadi beberapa sejumlah partisi tetap. Pada partisi-partisi tersebut proses-proses ditempatkan. Pemartisian statis berdasarkan ukuran partisi-partisinya terbagi dua, yaitu :

1. Pemartisian menjadi partisi-partisi berukuran sama, yaitu ukuran semua partisi memori adalah sama.

Beberapa proses yang ukurannya kurang atau sama dengan ukuran partisi dimasukkan ke sembarang partisi yang tersedia.

Kelemahan :

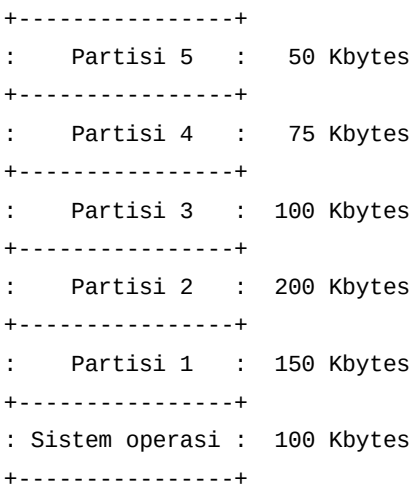
* Bila program berukuran lebih besar dibanding partisi yang tersedia, maka tidak dapat dimuatkan, tidak dapat dijalankan. Pemogram harus mempersiapkan overlay sehingga hanya bagian program yang benar-benar dieksekusi yang dimasukkan ke memori utama dan saling bergantian. Untuk overlay diperlukan sistem operasi yang mendukung swapping.
* Untuk program yang sangat kecil dibanding ukuran partisi yang ditetapkan, maka banyak ruang yang tak dipakai yang diboroskan, disebut fragmentasi internal.

Kelemahan ini dapat dikurangi dengan partisi-partisi tetap berukuran berbeda.

2. Pemartisian menjadi partisi-partisi berukuran berbeda, yaitu ukuran semua partisi memori adalah berbeda.

Gambar 6.5 menunjukkan skema multiprogramming pemartisian tetap berukuran

berbeda.



Gambar 6.5 : Multiprogramming dengan pemartisian tetap berukuran sama

6.8. Strategi penempatan program ke paritisi

a. Strategi penempatan pada pemartisian menjadi partisi-partisi berukuran sama. Penempatan proses ke memori dilakukan secara mudah karena dapat dipilih sembarang partisi yang kosong.
b. Strategi penempatan pada pemartisian menjadi partisi-partisi berukuran berbeda.

Terdapat dua strategi penempatan program ke partisi, yaitu :

a. Satu antrian untuk tiap partisi (banyak antrian untuk seluruh partisi).

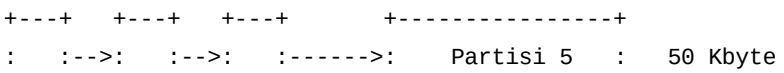
Proses ditempatkan ke partisi paling kecil yang dapat memuatnya.

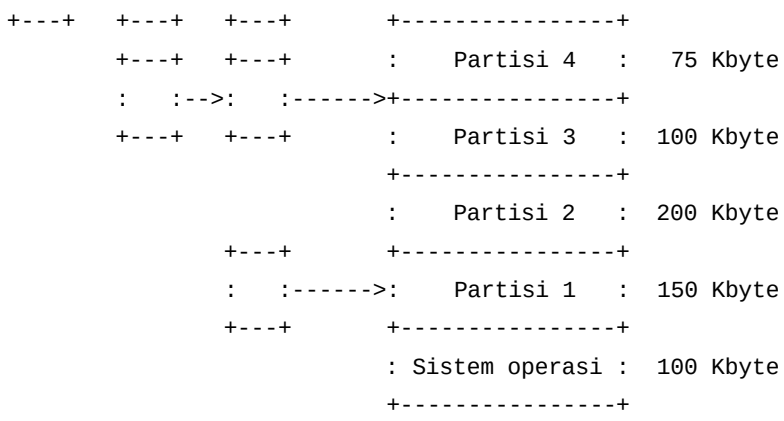
Keuntungan :

Teknik ini adalah meminimalkan pemborosan memori.

Kelemahan :

Dapat terjadi antrian panjang disuatu partisi sementara antrian partisi-partisi lain kosong. Teknik ini diperlihatkan pada gambar 6.6.





Gambar 6.6 : Multiprogramming dengan pengisian pemartisian tetap dengan banyak antrian.

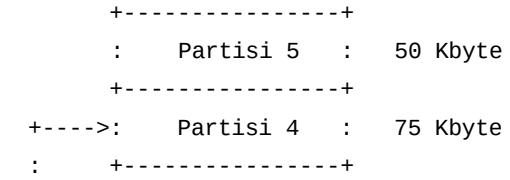
b. Satu antrian untuk seluruh partisi.

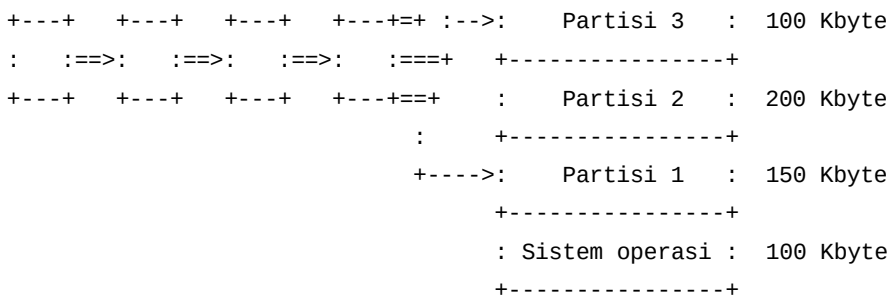
Proses-proses diantrikan di satu antrian tunggal untuk semua partisi.
Proses segera ditempatkan di partisi bebas paling kecil yang dapat memuat.

Keunggulan :
Lebih fleksibel serta implementasi dan operasi lebih minimal karena hanya mengelola satu antrian.

Kelemahan :
Proses dapat ditempatkan di partisi yang banyak diboroskan, yaitu proses kecil ditempatkan di partisi sangat besar.

Teknik ini diperlihatkan pada gambar 6.7.





Gambar 6.7 : Multiprogramming dengan pengisian pemartisian tetap dengan satu antrian.

Kelemahan ini dapat diatasi dengan prosedur pemindahan. Pemindahan dilakukan bila proses besar akan masuk memori tetapi hanya tersedia partisi kecil sementara proses kecil menempati partisi besar. Proses kecil di swap ke partisi kecil yang sedang bebas kemudian proses besar di antrian menempati partisi besar yang ditinggal proses kecil.

Pemartisian memori menjadi partisi-partisi secara statis mempunyai dua masalah, yaitu :

a. Relokasi.

Adalah masalah penempatan proses sesuai alamat fisik sehubungan alamat partisi memori dimana proses ditempatkan. Proses dapat ditempatkan pada partisi-partisi berbeda menurut keadaan sistem saat itu. Pengalamatan fisik secara absolut untuk proses tidak dapat dilakukan.

Solusi pertama :

Sistem operasi menambahkan alamat awal partisi dimana proses ditempatkan ke setiap alamat yang diacu proses. Pada saat proses kompilasi, linker harus memasukkan satu daftar atau bit map biner pada program memberitahu word program yang alamat-alamatnya direlokasi. Linker harus mencatat opcode, konstanta, dan item-item yang tak perlu direlokasi.

Masalah yang ditimbulkan :

Solusi ini menimbulkan masalah proteksi terhadap memori. Program tak terkendali selalu mampu membangun instruksi baru dan meloncati.

Tak ada cara untuk menghentikan jika program membaca atau menulis word di memori partisi lain (yang bukan hak-nya). Masalah relokasi dan proteksi tidak dapat dipisahkan, diperlukan satu solusi tunggal mengatas kedua masalah tersebut.

b. Proteksi.

Masalah proteksi pada banyak partisi dengan banyak proses di satu sistem secara bersamaan dikhawatirkan proses menggunakan atau memodifikasi daerah yang dikuasai proses lain (yang bukan haknya). Bila kejadian ini terjadi, maka proses lain dapat terganggu dan hasil yang diperolehnya dapat menjadi kacau.

Solusi IBM 360 :

Pada komputer IBM 360 membagi memori menjadi blok-blok, tiap blok ditambahi 4 bit kode proteksi. Blok berukuran 2 Kb. Proses jua mempunyai PSW (Program Status Word) yang antara lain berisi status proteksi. Status proteksi ini terdiri dari 4 bit (sama dengan bit kode proteksi untuk blok memori), merupakan kunci dalam pengaksesan memori. Proses hanya diijinkan mengakses blok-blok memori yang berkode proteksi sama dengan kode proteksi yang dimiliki PSW proses. Jika proses mengakses blok memori berkode proteksi berbeda dengan kunci PSW-nya, terjadi trap. Trap ini memberitahu sistem operasi bahwa telah terjadi pelanggaran memori, yaitu terdapat pengaksesan ke blok memori yang bukan wewenang proses yang menyebabkan trap.

Solusi menggunakan base register dan limit register :

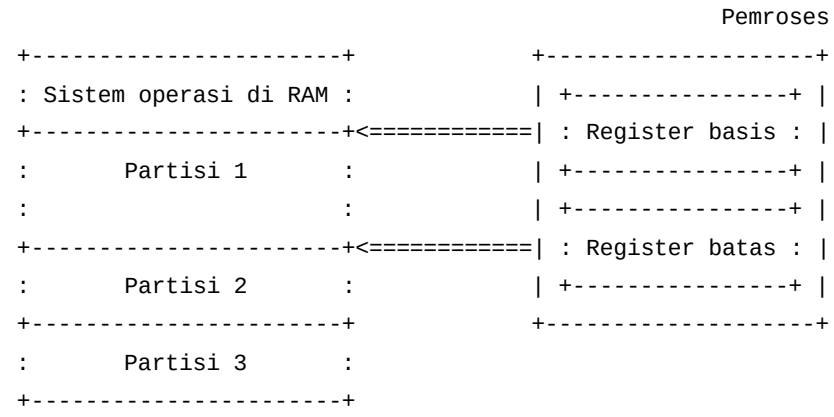
Solusi lain adalah menggunakan dua register yaitu base register dan limit register. Base register diisi alamat awal partisi dan limit register diisi panjang partisi. Setiap alamat yang dihasilkan secara otomatis ditambah dengan nilai base register. Instruksi yang mengacu pada alamat yang melebihi limit register akan menimbulkan trap yang memberitahu sistem operasi bahwa telah terjadi pelanggaran pengaksesan memori.

Teknik ini lebih unggul dibanding teknik pada IBM 360 karena sangat lebih efisien. Teknik ini tidak perlu menempatkan 4 bit proteksi di tiap blok memori. Teknik inipun lebih fleksibel.

Keuntungan :

a. Alamat tidak perlu dimodifikasi.
b. Setiap instruksi dapat diperiksa agar tidak meloncati batas limit register.
c. Program dapat dipindah walau sedang dieksekusi.
   Pemindahan dilakukan hanya dengan mengganti nilai base register.

Gambar 6.8 menunjukkan skema proteksi dan relokasi menggunakan register basis dan register batas. Register basis menunjuk alamat awal proses sedang register batas menunjuk alamat akhir proses. Bila proses mengacu alamat lebih dari alamat yang ditunjuk register batas maka pemroses mengirim sinyal fault yang memberitahu terjadinya pelanggaran pengaksesan memori.



Gambar 6.8 : Skema relokasi dan proteksi menggunakan register basis dan register batas.

6.9. Fragmentasi pada pemartisian statis.

Fragmentasi yaitu penyiaan/pemborosan memori akan terjadi pada setiap organisasi penyimpanan.

Fragmentasi pada pemartisian tetap terjadi adalah :

a. Fragmentasi internal.
   Proses tidak mengisi penuh partisi yang telah ditetapkan untuk proses.
b. Fragmentasi ekstenal.
   Partisi dapat tidak digunakan karena ukuran partisi lebih kecil dibanding ukuran proses yang menunggu di antrian, sehingga tidak digunakan.

Untuk sistem-sistem tanpa swapping (pemindahan lokasi proses), maka fragmentasi-fragmentasi tidak dapat dikurangi. Pada sistem-sistem dengan swapping, sistem lebih intelijen karena dapat melakukan beberapa altenatif mengatasi fragmentasi eksternal.

6.10. Multiprogramming dengan swapping

Pada sistem batch, organisasi memori dengan pemartisian tetap telah efektif. Selama jumlah proses yang terseduan dapat membuat pemroses sibuk, tak ada alasaan menggunakanan teknik lebih rumit. Pada sistem timesharing, situasinya berbeda, umumnya terdapat lebih banyak proses dibanding memori yang tersedia untuk memuat seluruh proses. Dengan demikian perlu menyimpan proses-proses yang tidak termuat ke disk. Untuk menjalankan proses-proses yang akan dieksekusi, proses-proses itu harus telah masuk memori utama.

Pemindahan proses dari memori utama ke disk dan sebaliknya di sebut swapping. Dengan swapping, multiprogramming pada sistem time sharing dapt ditingkatkan kinerjanya yaitu dengan memindah proses-proses blocked ke disk dan hanya memasukkkan proses-proses ready ke memori utama. Beragam masalah harus diatasi multiprogramming dengan swapping, antara lain :

a. Pemartisian secara dinamis.
b. Strategi pencatatan pemakaian memori.
c. Algoritma penempatan proses ke memori.
d. Strategi penempatan ruang swap pada disk.

6.11. Multiprogramming dengan pemartisian dinamis

Pemartisian statis tidak menarik karena terlalu banyak diboroskan proses-proses yang lebih kecil dibanding partisi yang ditempatinya. Dengan pemartisian dinamis maka jumlah, lokasi dan ukuran proses di memori dapat beragam sepanjang waktu secara dinamis. Proses yang akan masuk ke memori segera dibuatkan paritisi untuknya sesuai kebutuhannya. Teknik ini meningkatkan utilitasi memori.

Kelemahan pemartisian dinamis adalah :

a. Dapat terjadi lubang-lubang kecil memori di antara partisi-partisi yang dipakai.

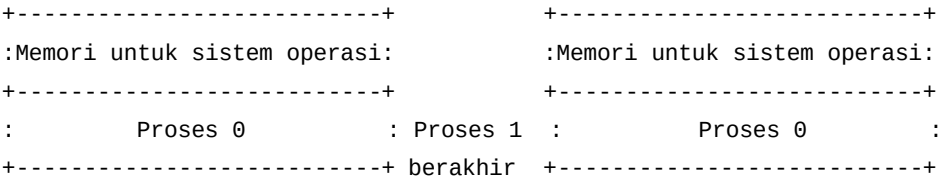
b. Merumitkan alokasi dan dealokasi memori.

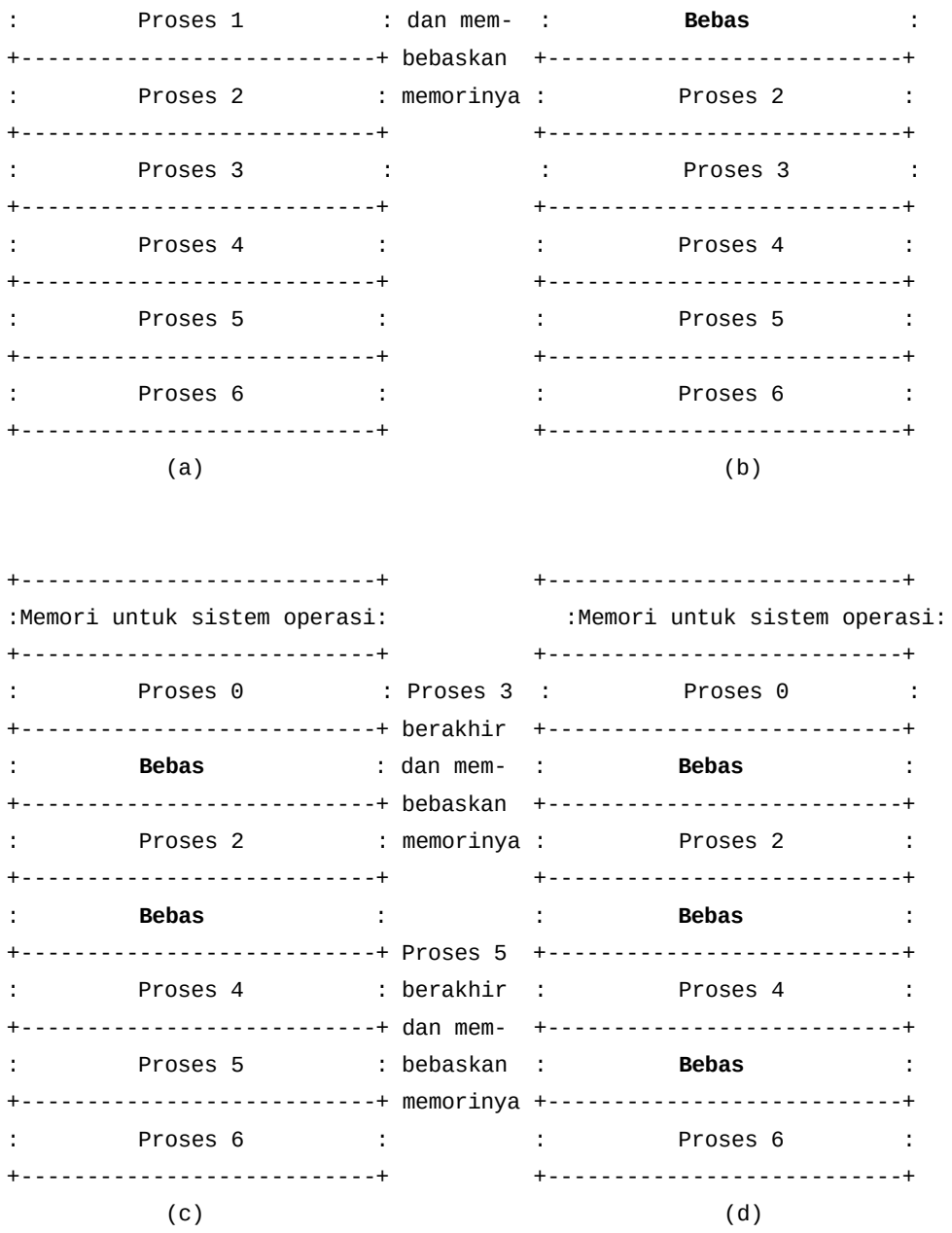
6.12. Terjadi lubang-lubang kecil memori

Contoh terjadinya lubang-lubang di antara partisi-partisi adalah gambar 6.9 :

* Gambar 6-9(a) adalah konfigurasi awal, terdapat 7 proses di ruang memori. Setelah proses 1 berakhir, konfigurasi memori menjadi gambar 6-9(b).
* Begitu proses 3 berakhir, konfigurasi memori menjadi gambar 6-9(c).
* Proses 5 berakhir, menghasilkan konfigurasi gambar 6-9(d). Memori dipenuhi lubang-lubang memori yang tak terpakai.

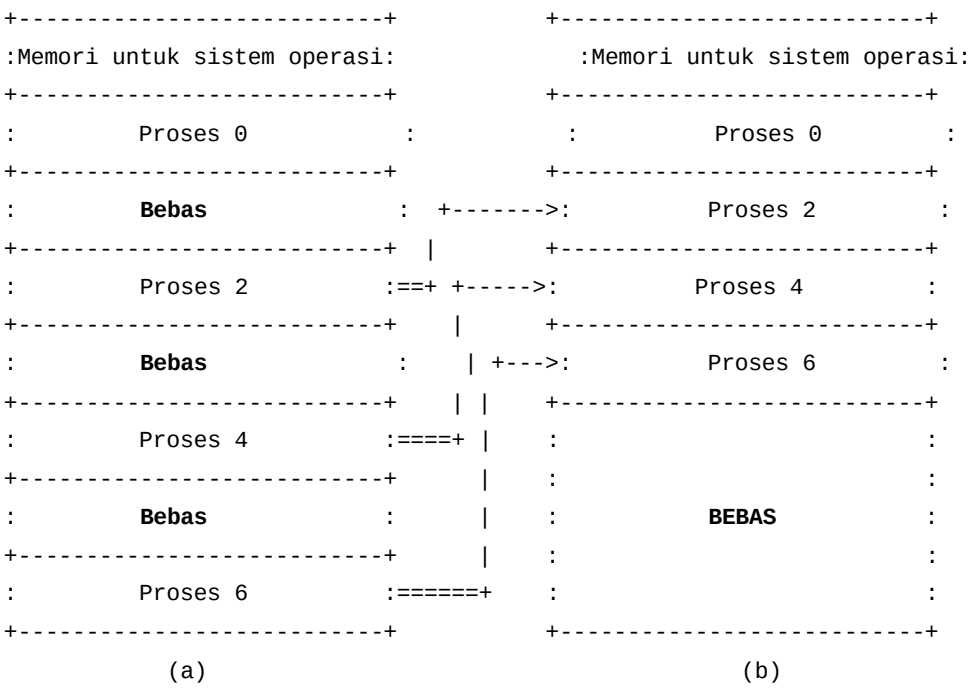
Lubang-lubang (yaitu kelompok blok-blok memori yang tidak digunakan) kecil di antara blok-blok memori yang digunakan dapat diatasi dengan pemadatan memori (memory compaction). Pemadatan memori adalah operasi menggabungkan semua lubang kecil menjadi satu lubang besar dengan memindahkan semua proses agar saling berdekatan.





Gambar 6.9 : Alokasi memori secara dinamis.

Gambar 6-10 menunjukkan skema pemadatan memori. Proses 2, 4, dan 6 dipindahkan agar menampati ruang-ruang berturutan dengan proses 0 sehingga diperoleh lubang memori besar. Lubang memori besar ini dapat ditempati proses yang akan masuk.



Gambar 6.10 : Lubang-lubang memori dan pemadatan memori.

Kelemahan utama teknik pemadatan memori :

- ☑ Memerlukan waktu yang sangat banyak.
- ☑ Sistem harus menghentikan sementara semua proses selagi melakukan pemadatan. Hal in meningkatkan waktu tanggapan di sistem interaktif dan tak mungkin digunakan di sistem waktu nyata real.

6.13. Proses tumbuh berkembang

Masalah lain pada pemartisian dinamis adalah proses dapat tumbuh berkembang.

Segmen data proses dapat tumbuh, misalnya karena :

* Heap untuk data dinamis berkembang.

* Stack untuk pemanggilan prosedur dan variabel lokal.

Solusi masalah ini adalah bila proses bersebelahan dengan lubang memori tak dipakai, proses tumbuh memakai lubang itu. Masalah menjadi parah bila proses bersebelahan dengan proses-proses lain.

Peringkat alternatif penyelesaian adalah :

- ☑ Bila masih terdapat lubang besar yang dapat memuat proses, maka proses dipindah ke lubang memori yang cukup dapat memuat.
- ☑ Satu proses atau lebih di swap ke disk agar memberi lubang cukup besar untuk proses yang berkembang.
- ☑ Jika proses tidak dapat tumbuh di memori dan daerah swap di disk telah penuh, proses harus menunggu atau disingkirkan.

6.14. Pencatatan pemakain memori

Memori yang tersedia harus dikelola, dilakukan dengan pencatatan pemakaian memori. Terdapat dua cara utama pencatatan pemakaian memori, yaitu :

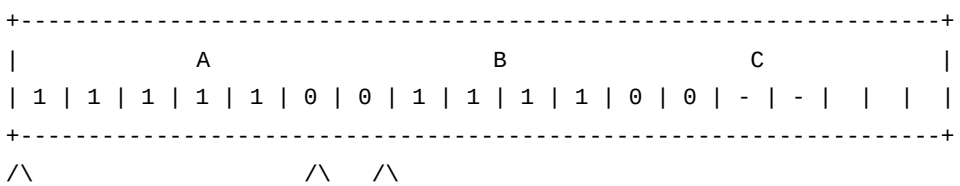
1. Pencatatan memakai peta bit.

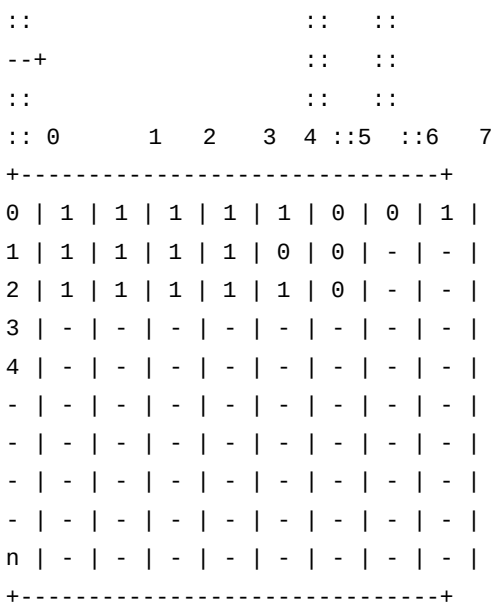
Memori dibagi menjadi unit-unit alokasi,berkorespondensi dengan tiap unit alokasi adalah satu bit pada bit map.

* Nilai 0 pada peta bit berarti unit itu masih bebas.

* Nilai 1 berarti unit digunakan.

Gambar 6-11 menunjukkan skema peta bit untuk pencatatan pemakaian memori. Elemen peta bit bernilai 1 menunjukkan blok tersebut telah digunakan oleh proses dan bernilai 0 yang berarti belum digunakan oleh proses.





Gambar 6-11 : Peta bit untuk pengelolaan pemakaian memori.

Masalah

Masalah pada peta bit adalah penetapan mengenai ukuran unit alokasi memori, yaitu :

- ☑ Unit lokasi memori berukuran kecil berarti membesarkan ukuran peta bit.
- ☑ Unit alokasi memori n berukuran besar berarti peta bit kecil tapi memori banyak disiakan pada unit terakhir jika ukuran proses bukan kelipatan unit alokasi.

Keunggulan :

* Dealokasi dapat dilakukan secara mudah, hanya tinggal menset bit yang berkorespondensi dengan unit yang telah tidak digunakan dengan 0.

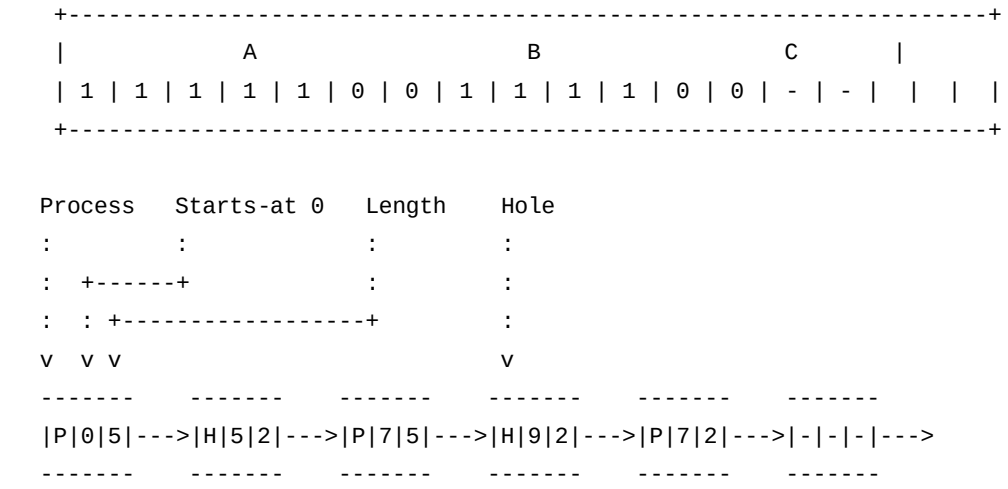
Kelemahan :

* Harus dilakukan penghitungan blok lubang memori saat unit memori bebas.
* Memerlukan ukutan bit map besar untuk memori yang besar.

2. Pencatatan memakai senarai berkait.

Sistem operasi mengelola senarai berkait (linked list) untuk segmen-segmen memori yang telah dialokasikan dan bebas. Segmen memori menyatakan memori

untuk proses atau memori yang bebas (lubang). Senarai segmen diurutkan sesuai alamat blok.



Gambar 6-12 : Pengelolaan pemakaian memori dengan senarai berkait.

Keunggulan :

* Tidak harus dilakukan perhitungan blok lubang memori karena sudah tercatat di node.
* Memori yang diperlukan relatif lebih kecil.

Kelemahan :

* Dealokasi sulit dilakukan karena terjadi berbagai operasi penggabungan node-nude di senarai.

6.15. Strategi alokasi memori

Terdapat berbagai strategi alokasi proses ke memori. Alokasi harus mencari sekumpulan blok memori yang ukurannya mencukupi memuat proses yaitu lubang kosong yang sama atau lebih besar dibanding ukuran memori yang diperlukan proses.

Beragam algoritma itu antara lain :

*** First-fit algorithm.**

Strategi in dapat dilakukan pada pencatatan memori dengan bit map maupun senarai berkait. Manajer memori menscan sampai menemukan lubang besar yang mencukupi penempatan proses. Lubang dibagi dua, untuk proses dan lubang tak digunakan, kecuali ketika lubang tersebut tepat sama dengan ukuran yang diperlukan proses.

Keunggulan :

- Algoritma ini akan menemukan lubang memori paling cepat dibanding algoritma-algoritma lain.

*** Next-fit algorithm.**

Strategi ini dapat dilakukan pada pencatatan memori dengan bit-map maupun senarai berkait. Mekanisme algoritma ini sama dengan algoritma first fit algorithm, hanya tidak dimulai di awal tapi dari posisi terakhir kali menemukan segmen paling cocok.

Simulasi oleh Bays (1977) menunjukkan next-fit algorithm berkinerja lebih buruk dibanding first-fit algorithm.

*** Best-fit algorithm.**

Strategi ini dapat dilakukan pada pencatatan memori dengan bit-map maupun senarai berkait. Algoritma mencari sampai akhir dan mengambil lubang terkecil yang dapat memuat proses. Algoritma ini mencoba menemukan lubang yang mendekati ukuran yang diperlukan.

*** Worst-fit algorithm.**

Strategi ini dapat dilakukan pada pencatatan memori dengan bit-map maupun senarai berkait. Selalu mencari lubang besar yang tersedia sehingga lubang dapat dipecah menjadi cukup besar, agar berguna untuk proses-proses berikutnya. Simulasi menunjukkan worst-fit algorithm bukan gagasan yang bagus.

*** Quick-fit algorithm.**

Strategi ini hanya untuk pencatatan memori dengan senarai berkait.

Keempat algoritma dapat dipercepat dengan mengelola dua senarai, yaitu :

- Senarai untuk proses.
- Senarai untuk lubang memori.

Dengan cara ini, saat alokasi hanya perlu menginspeksi senarai lubang, tidak perlu senarai proses.

Keunggulan :

· Teknik ini mempercepat pencarian lubang atau penempatan proses.

Kelemahan :

· Kompleksitas dealokasi memori bertambah dan melambatkan dealokasi memori karena memori yang dibebaskan harus dipindahkan dari senarai proses ke senarai lubang.

*** Quick fit.**

Cara diatas dapat diperluas, algoritma mengelola sejumlah senarai lubang memori dengan beragam ukuran yang paling sering diminta.

Contoh :

Algoritma mengelola senarai lubang sebagai berikut :

· Senarai 8 Kb.
· Senarai 12 Kb.
· Senarai 20 Kb.
· Senarai 40 Kb.
· Senarai 60 Kb.
· Dan seterusnya.

Senarai mencatat lubang-lubang memori sesuai ukuran lubang. Lubang-lubang memori dimuat di senarai sesuai ukuran terdekat, misalnya lubang memori 42 dimuat pada senarai 40 Kb. Dengan beragam senarai maka alokasi memori dapat dilakukan dengan cepat yaitu tinggal mencarai senarai terkecil yang dapat menampung proses tersebut.

Keunggulan :

· Algoritma ini sangat cepat dalam alokasi proses.

Kelemahan :

· Dealokasi sulit dilakukan.

Ketika proses berakhir atau dipindah keluar (swap-out) maka menemukan tetangga-tetangga memori yang dipakai proses untuk penggabungan adalah

sangat mahal/lama. Jika penggabungan tidak dilakukan, memori akan segera menjadi banyak lubang kecil yang tak berguna.

6.16. Sistem Buddy

Sistem buddy adalah algoritma pengelolaan memori yang memanfaatkan kelebihan penggunaan bilangan biner dalam pegalamatan memori. Karakteristik bilangan biner digunakan untuk mempercepat penggabungan lubang-lubang berdekatan ketika proses terakhir atau dikeluarkan.
Manajer memori mengelola senarai blok-blok bebas berukuran 1, 2, 4, 8, 16 byte dan seterusnya sampai kapasita memori. Pada komputer dengan 1 Mbyte memori maka dapat terdapat 21 senarai yaitu dari 1 byte sampai 1 Mbyte.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	lubang
Semula																	1
Minta 85 kb	A																-
Minta 45 kb	A	B															-
Minta 75 kb	A	B	C														-
A didealokasi		B	C														-
Minta 55 kb		B	D	C													-
B didealokasi			D	C													-
D didealokasi				C													-
C didealokasi																	-

Gambar 6.13 : Pengelolaan memori dengan sistem Buddy.

Mekanisme pengelolaan :

- Awalnya semua memori adalah bebas dan hanya satu senarai 1 Mbyte yang terisi berisi satu isian tunggal satu lubang 1 Mbyte. Senarai-senarai lain adalah kosong.

Misalnya proses baru berukuran 85 Kbyte mekanisme yang dijalankan adalah sbb :

- ☑ Karena hanya terdapat senarai berisi 2k, maka permintaan 85 kb dialokasikan ke yang terdekat yaitu berarti 128 kb, 2k terkecil yang mampu memuat.
- ☑ Karena tidak tersedia blok berukuran 128 kb, atau 256 kb atau 512 kb, maka blok 1 Mb dipecah menjadi dua blok 512 kb. Blok-blok pecahan ini disebut buddies. Satu beralamat mulai dari 0 dan lainnya mulai alamat 512 k.
- ☑ Salah satu blok 512 kb yang beralamat 0 dipecah lagi menjadi dua blok buddies 256 kb. Satu beralamat mulai dari 0 dan lainnya mulai alamat 256 kb.
- ☑ Blok 256 pada alamat 0, kemudian dipecah menjadi 2 blok buddies 128 kb.
- ☑ Blok yang pertama dialokasikan ke proses yang baru.

Keunggulan :

- ☑ Sistem buddy mempunyai keunggulan dibanding algoritma-algoritma yang mengurutkan blok-blok berdasarkan ukuran. Ketika blok berukuran 2k dibebaskan, maka manajer memori hanya mencari pada senarai lubang 2k untuk memeriksa apakah dapat dilakukan penggabungan. Pada algoritma-algoritma lain yang memungkinkan blok-blok memori dipecah dalam sembarang ukuran, seluruh senarai harus dicari.
- ☑ Dealokasi pada sistem buddy dapat dilakukan dengan cepat.

Kelemahan :

· Utilisasi memori pada sistem buddy sangat tidak efisien.

Masalah ini muncul dari dari kenyataan bahwa semua permintaan dibulatkan ke 2k terdekat yang dapat memuat. Proses berukuran 35 kb harus dialokasikan di 64 kb, terdapat 29 kb yang disiakan. Overhead ini disebut fragmentasi internal karena memori yang disiakan adalah internal terhadap segmen-segmen yang dialokasikan.

6.17. Alokasi ruang swap pada disk

Strategi dan algoritma yang dibahas adalah untuk mencatat memori utama.
Ketika proses akan dimasukkan ke memori utama (swap-in), sistem dapat menemukan ruang untuk proses-proses itu.

Terdapat dua strategi utama penempatan proses yang dikeluarkan dari memori utama (swap-out) ke disk, yaitu :

- **Ruang disk tempat swap dialokasikan begitu diperlukan.**

  Ketika proses harus dikeluarkan dari memori utama, ruang disk segera dialokasikan sesuai ukuran proses. Untuk itu diperlukan algoritma untuk mengelola ruang disk seperti untuk mengelola memori utama. Ketika proses dimasukkan kembali ke memori utama segera ruang disk untuk swap didealokasikan.

- **Ruang disk tempat swap dialokasikan lebih dulu.**

  Saat proses diciptakan, ruang swap pada disk dialokasikan. Ketika proses harus dikeluarkan dari memori utama, proses selalu ditempatkan ke ruang yang telah dialokasikan, bukan ke tempat-tempat berbeda setiap kali terjadi swap-out. Ketika proses berakhir, ruang swap pada disk didealokasikan.