Universitas Esa Unggul Kampus Harapan Indah	**MODUL 9** **CCS 210 SISTEM OPERASI**	
Judul	MANAJEMEN MEMORY	
Penyusun	**Distribusi**	**Perkuliahan**
Nixon Erzed	**FASILKOM** **UNIVERSITAS ESA UNGGUL**	Pertemuan OL – 10

Materi

- Konsep dasar manajemen memory
- Binding
- Swapping dan Fragmentasi
- Teknik Pemartisian

Tujuan

Mahasiswa memahamai konsep dasar pengelolaan memory

DASAR MANAJEMEN MEMORY

Latar Belakang

- Memori adalah pusat kegiatan pada sebuah komputer → setiap proses yang akan dijalankan, harus telah berada di memori.
- CPU mengambil **instruksi** dari memori sesuai alamat yang ada pada *Program Counter (pengindentifikasian ruang alamat → pola terurut (sequensial) → proses harus berada dalam suatu block kontigue)*

  Instruksi dapat berupa
  - menyimpan dari/ke alamat di memori,
  - penambahan, dsb.

Dalam konteks multitasking

- Tugas sistem operasi adalah mengatur peletakan banyak proses pada suatu memori.
- Memori harus dapat digunakan dengan baik, sehingga dapat memuat banyak proses dalam suatu waktu.

Mengapa Manajemen Memory (latar belakang teknis → kasus)

1. Memory Utama adalah sumber daya yang harus dialokasikan dan dipakai bersama diantara sejumlah proses yang aktif.
   → Untuk mencapai efisiensi pemanfaatan pemroses dan fasilitas I/O, maka mesti diupayakan agar memory dapat menampung sebanyak mungkin proses.
2. Kadang-kadang program melebihi kapasitas memory

Tujuan manajemen memory : → kriteria yang harus dicapai

1. meningkatkan utilitas CPU sebesar-besarnya
   semakin banyak proses yang aktif dalam sistem → interleave proses-proses lebih dapat dijamin→ sistem selalu sibuk (layanan proses) → utilitas CPU relatif lebih baik

2. data dan instruksi dapat diakses dengan cepat oleh CPU
   penempatan data/instruksi dalam memory harus menjamin kemudahan untuk diakses

3. Memory utama memiliki kapasitas yang sangat terbatas sehingga pemakaiannya harus seefisien mungkin → meminimalisasi fragmentasi

4. Transfer data dari/ke memory utama ke/dari CPU dapat seefisien mungkin

Kebutuhan pengelolaan memory antara lain : (fungsi yang harus tersedia)

a. Relokasi dan translasi
   Prosesor dan system operasi harus dapat mentranslasikan memory referensi (dalam bentuk kode program) ke alamat fisik yang mengalokasikan program dalam memory utama.

b. Proteksi
   User tidak boleh mengakses beberapa bagian dari wilayah memori (misal area system operasi/kernel → dengan memanfaatkan reg. basis dan limit)

c. Sharing
   Manajemen memory harus dapat mengontrol sharing area pada memory utama

d. Organisasi Logika
   Sistem operasi dan hardware diusahakan untuk dapat berhubungan dengan user program dalam satu modul→ layanan-layanan sistem (system call) tentang memory yang akan digunakan oleh User program → tidak dibedakan antara aktifitas HW dan logika OS

e. Organisasi Fisik
   Harus ada pengaturan yang jelas antara memory utama dengan memory sekunder (extended area) pada midle term scheduling

Misal terdapat program sebagai berikut :

```
        Inst  1
        Inst  2
        Jump kota
        Inst ..
        ..
        ..
kota    Inst 20
        inst 21
```

setelah dikompilasi→ setiap intsruksi memiliki alamat relatif

```
1       Inst  1          →  3200 H
2       Inst  2           3201
3       Jump 24          → jump 24  harus ditranslasikan menjadi jump 3218H
4       Inst ..
        ..
        ..
24      Inst XY                → 3218 H
        inst Z
```

ketika program dijalankan → program akan diload ke memory
→ alamat referensi harus ditranslasikan

Misalkan awal program tersebut ditempatkan di alamat 3200 H
Alamat referensi *24* harus ditranslasikan sesuai dengan lokasi/alamat fisik instruksi XY

PEMBERIAN ALAMAT
(TEKNIK BINDING)

Binding → menentukan ruang alamat (memory) yang akan diberikan/ditempati proses. Dalam pengertian umum, ruang alamat akan ditetapkan ketika proses akan dimuat (load) ke memory.

Terdapat 3 teknik binding yaitu :

1. Pengikatan Waktu Kompilasi (compile time binding)
2. Pengikatan Waktu Pemanggilan/Pemuatan (loading time binding)
3. Pengikatan Waktu Eksekusi (run time binding)

Waktu Kompilasi (compile time binding)

Ruang alamat yang akan ditempati oleh proses diketahui pada waktu kompilasi. Untuk kemudian kode absolutnya dapat dibuat.
→ Jika kemudian alamat awalnya berubah (referensi pengalamatan sistem HW), maka harus dikompilasi ulang.

Waktu Pemanggilan/Pemuatan (loading time binding)

→ ruang memory ditentukan ketika proses diciptakan
Jika tidak diketahui dimana poses ditempatkan di memori, maka kompilator harus membuat kode yang dapat dialokasikan. Dalam kasus pengikatan akan ditunda sampai waktu pemanggilan. Jika alamat awalnya berubah, kita hanya perlu menempatkan ulang kode, untuk menyesuaikan dengan perubahan.

Waktu Eksekusi (run time binding)

Jika proses dapat dipindahkan dari suatu segmen memori ke lainnya selama dieksekusi. Pengikatan akan diperbaharui pada *run-time*.

- akibat relokasi
- implementasi virtual memory
- resume proses suspended
- pengaktifan proses anak (?)

Skenario binding :

Sebelum masuk ke memori, suatu proses harus menunggu. Hal ini disebut antrian masukan. Proses-proses ini akan berada dalam beberapa tahapan sebelum dieksekusi.

- Alamat-alamat yang dibutuhkan mungkin saja direpresentasikan dalam cara yang berbeda dalam tahapan-tahapan ini.
- Alamat dalam kode program masih berupa simbolik.
- Alamat ini akan diikat oleh kompilator ke alamat memori yang dapat diakses.
- Kemudian *linkage editor* dan *loader*, akan mengikat alamat fisiknya.
- Setiap pengikatan akan memetakan suatu ruang alamat ke lainnya.

TEKNIK/PENDEKATAN DALAM PENEMPATAN PROSES

A. Dynamic Loading

Karena memory utama sangat terbatas, maka dimungkinkan hanya menempatkan bagian-bagian yang diperlukan saja yang harus tetap tinggal dalam memory. Dengan dynamic loading ini suatu rutin tidak akan diambil sampai rutin tersebut dibutuhkan.

Pada saat suatu rutin butuh memanggil rutin yang lainnya, maka pertama-tama rutin pemanggil tersebut memeriksa apakah rutin yang dibutuhkan tersebut sudah pernah diambil, jika belum maka rutin tersebut diambil dan dialokasikan di memory utama.

B. Overlay

Overlay adalah membagi program yang besar menjadi bagian yang kecil-kecil dan dapat dimuat dalam memory utama. Yang harus selalu ada dalam memory adalah bagian penggerak program. Bagian program lainnya tetap pada memory sekunder dan akan di upload jika diperlukan dan dikembalikan jika telah selesai digunakan.

C. Dynamic Linking

Mirip dengan dynamic loading+overlay, dengan penekanan proses linking terhadap library bersama. Dengan dynamic linking maka aplikasi tidak perlu menduplikasi library ke modul prosesnya, tapi library ditempatkan di area bersama

SWAPPING DAN PEMARTISIAN MEMORY

Mengalihkan/memindahkan sementara → misalnya untuk suatu proses yang tertunda (suspend)

- SWAP OUT : dari memori ke secondary storage (extended memory)
- SWAP IN : resume kembali proses ke memori → jika proses akan dilanjutkan.

Swapping diterapkan pada manajemen memori dengan pemartisian dinamis. Pada manajemen memori dengan pemartisian statis tidak memerlukan swapping.

Swapping diterapkan juga untuk relokasi proses (run time binding)
Persoalannya : relokasi/transformasi alamat

Ketika Swap In → proses tidak akan menempati kembali ruang alamat (lokasi memory) yang semula, sehingga berbagai pencatatan alamat referensi harus dapat direlokasi dan ditraslasikan

Dasar untuk pendekatan dalam pengelolaan ruang memory :

1. ruang-ruang alamat harus dapat diidentifikasi : terpakai atau tidak terpakai
2. indentifikasi berdasarkan per alamat → informasi identifikasi sangat besar
3. harus didefinisikan satuan ruang sebagai basis indentifikasi
   misal : 1 unit ruang = 100 KB

TEKNIK PEMARTISIAN Memory

Pemartisian (partisi) → membagi area memori atas satuan kelompok ruang

→ 1 proses akan menempati 1 partisi

→ 1 partisi akan ditempati 1 proses

Tujuan :
untuk penstrukturan pengelolaan sistem memory → menghidarkan kerumitan

Teknik pemartisian → 2 type

1. Partisi statis
   Ukuran partisi ditetapkan saat pemrograman sistem (bersifat tetap)
   → Partisi berukuran sama
   → Partisi berukuran beraneka

2. Partisi Dinamis

   Pada awalnya → seluruh ruang didefinisikan sebagai 1 partisi besar

   Setiap datang sebuah proses → akan dibuat sebuah partisi dengan ukuran seduai kebutuhan

   Jika sebuah proses selesai → maka bekas ruang yang ditempatinya dileburkan kembali ke partisi besar

FRAGMENTASI

Fragmentasi : adalah sisa ruang memori yang tidak dapat dialokasikan untuk suatu proses.

Ada dua fragmentasi :

1. Fragmentasi internal,

   Proses tidak mengisi secara penuh partisi/unit yang dialokasikan, akibat ukuran partisi/unit lebih besar dari ukuran proses.

2. Fragmentasi eksternal

   Partisi/kumpulan unit yang kontigue tidak dapat digunakan karena ukuran partisi/kumpulan unit lebih kecil dari ukuran proses yang akan masuk.

TEKNIK PEMARTISIAN STATIS
(manajemen memory tanpa swapping)

Terdapat dua bentuk manajemen memori tanpa swapping :

- Monoprogramming
- Multiprogramming dengan pemartisian statis

1. Monoprogramming

→ hanya mengijinkan satu pemakai/program yang berjalan pada satu waktu.

→ Semua sumberdaya sepenuhnya dikuasai proses yang sedang berjalan.

Ciri-cirinya :

- Hanya satu proses pada satu saat
- Semua memori (jatah proses) dialokasikan untuk satu proses tersebut (diluar kernel)
- Semua program diload kememori
- Program mengambil kendali seluruh system
- Tidak memerlukan Swapping

Implementasi proteksi :

→ Pemakai mempunyai kendali penuh terhadap seluruh memori utama.

→ Proteksi rutin system operasi agar terjamin dari penghancuran oleh program pemakai.

- Proteksi diterapkan menggunakan suatu register batas (base & Limit Register) di pemroses.
- Setiap kali program mengacu alamat memori dibandingkan dengan register batas, untuk memastikan proses pemakai tidak merusak rutin sistem operasi.

2. Multiprogramming dengan pemartisian statis

Dalam implementasi multiprogramming, memori dibagi menjadi sejumlah partisi tetap. Proses-proses akan menempati partisi-partisi tersebut.

Pemartisian memori menjadi partisi-partisi yang berukuran sama :

→ Sebuah program akan menempati sebuah partisi,

- jika program berukuran lebih besar dari partisi → tidak dapat dimuat → program tidak dapat dieksekusi
- jika program berukuran jauh lebih kecil → terjadi ruang sisa yang tidak dapat dipakai oleh proses lain (fragmentasi internal).

Kelemahan pemartisian partisi statis tersebut diatasi :

→ membuat partisi-partisi yang berukuran berbeda.

Strategi penempatan Program ke Partisi

a. Penempatan pada partisi berukuran sama
   Dilakukan secara mudah, karena dapat dipilih sembarang partisi yang kosong

b. Penempatan pada partisi berukuran berbeda
   Terdapat dua strategi penempatan program pada partisi yaitu :
   - Satu antrian untuk seluruh partisi ***(satu antrian).***
     Proses segera ditempatkan pada partisi bebas paling kecil yang dapat memuatnya.
   - Satu antrian untuk sekelompok partisi **(banyak antrian)**
     Proses akan diantrikan pada partisi yang paling pas untuknya.
     Terdapat kemungkinan terjadi antrian yang panjang pada sebuah kelompok partisi.

TEKNIK PEMARTISIAN DINAMIS
(MANAJEMEN MEMORI DENGAN SWAPPING)

→ partisi didefinisikan ketika data/sebuah proses akan diloading ke memory

Resiko fragmentasi internal → lebih akibat ukuran satuan ruang /unit

Pada system dengan time sharing terdapat lebih banyak proses dibandingkan memori yang tersedia.

→ diperlukan pemindahan proses-proses yang sedang tidak running dari memori ke disk (swapping).

Dengan swapping, multitasking pada time sharing system dapat ditingkatkan kinerjanya yaitu dengan memindahkan proses-proses yang blocked ke disk dan hanya menempatkan proses-proses ready di memory.

Implementasi swapping → menimbulkan masalah relokasi alamat

IMPLEMENTASIKAN TEKNIK PEMARTISIAN DINAMIS

→ Multitasking dengan Pemartisian Dinamis

Masalah partisi statis

→ terjadi pemborosan memori karena fargmentasi yang muncul akibat penempatan proses berukuran lebih kecil dibanding partisi yang ditempatinya

Solusi → Partisi dinamis

→ Dengan pemartisian dinamis maka jumlah, lokasi, dan ukuran partisi di memori dapat beragam sepanjang waktu secara dinamis. Proses yang akan masuk ke memori segera dibuatkan partisi untuknya sesuai kebutuhannya.

Persoalan yang pada pemartisian dinamis :

- Munculnya lubang-lubang kecil di antara partisi-partisi yang dipakai

- Proses Tumbuh Berkembang
- Alokasi dan dealokasi rumit

1. Munculnya lubang-lubang kecil di antara partisi-partisi yang dipakai

   Pembuatan partisi akan dilakukan jika ada proses yang masuk. Jika suatu proses sudah menyelesaikan prosesnya maka memori akan di-dealokasi, sehingga memunculkan lubang memori. Jika ada proses baru yang akan masuk akan diperiksa apakah terdapat area kosong yang dapat memuat proses tersebut, misalkan ukuran proses yang baru sedikit lebih kecil dibanding proses yang sudah selesai, maka akan dibuatkan partisi baru dibekas proses yang sudah selesai. Akibatnya terdapat sisa area memori yang bersifat bebas, tapi berukuran sangat kecil yang sulit dialokasikan untuk suatu proses.

2. Proses Tumbuh Berkembang

   Selama pelaksanaan proses, terdapat kemungkinan ukuran proses dan data bertambah, misalnya pada penggunaan stack atau linked list.

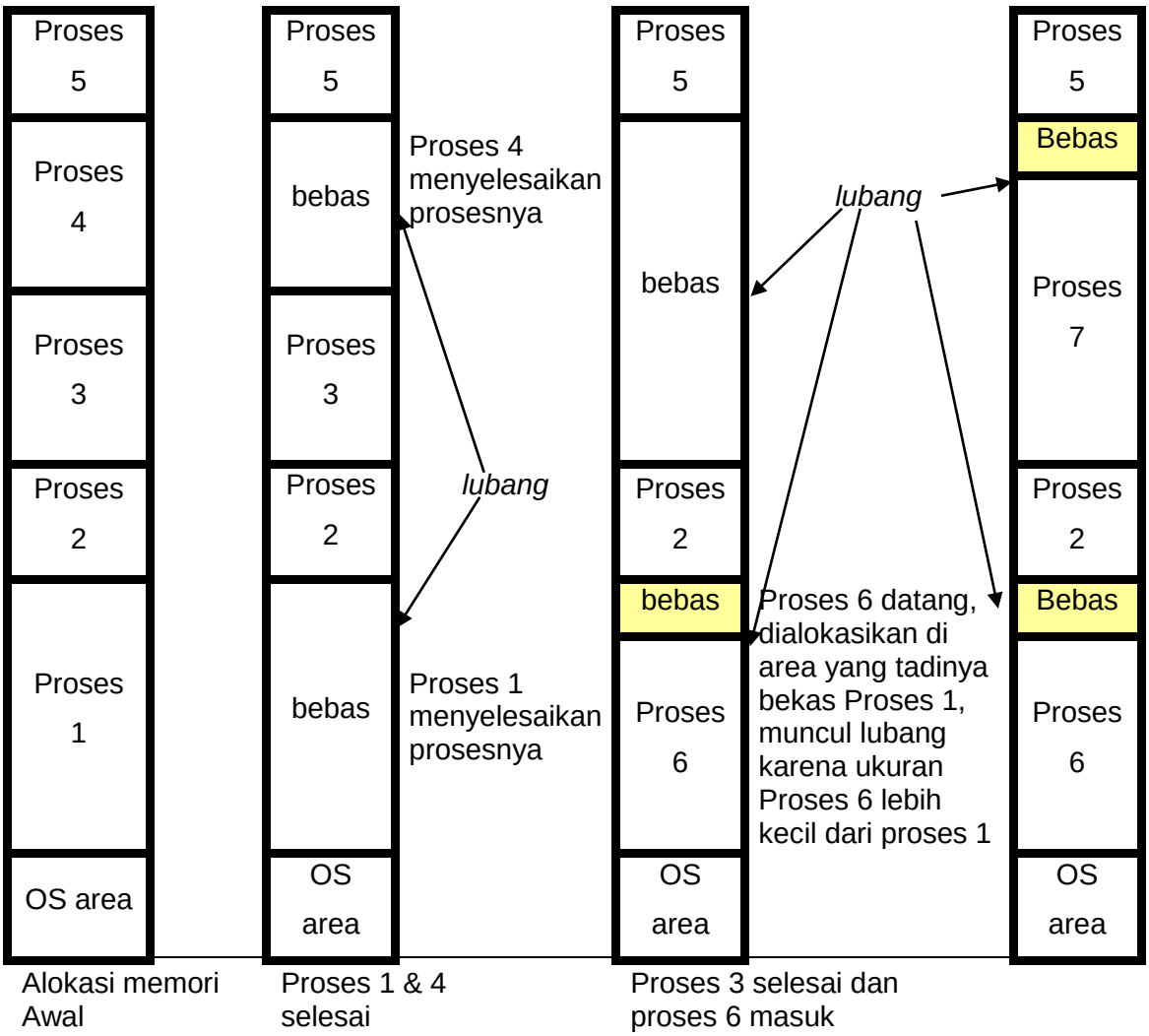
3. Alokasi dan dealokasi rumit

   Dalam menentukan area memori yang akan ditempati proses harus dilakukan pemeriksaan semua area kosong yang mungkin dan paling sesuai untuk proses tersebut. Diperlukan pencatatan pemakaian memori dan strategi alokasi yang paling efektif.

Solusi

1. fragmentasi eksternal → defragmentasi / pemadatan
   - rumit → karena implementasi defragmentasi membutuhkan penghentian seluruh proses (kecuali kernel)
   - defragmentasi tidak dapat diterapkan jika ada proses yang bersifat Real Time
   - pendekatan defragmentasi → suspend seluruh proses dan resume secara bertahap

2. relokasi proses menggunakan mekanisme suspend proses


Proses 5
Proses 4
Proses 3
Proses 2
Proses 1
OS area
Alokasi memori Awal
Proses 5
bebas
Proses 3
Proses 2
bebas
OS area
Proses 1 & 4 selesai
Proses 4 menyelesaikan prosesnya
lubang
Proses 1 menyelesaikan prosesnya
Proses 5
bebas
Proses 2
bebas
Proses 6
OS area
Proses 3 selesai dan proses 6 masuk
lubang
Proses 6 datang, dialokasikan di area yang tadinya bekas Proses 1, muncul lubang karena ukuran Proses 6 lebih kecil dari proses 1
Proses 5
Bebas
Proses 7
Proses 2
Bebas
Proses 6
OS area

Pertimbangan dalam memilih Strategi Manajemen Memory

Beberapa pertimbangan dalam menentukan strategi manajemen memory antara lain adalah :

1. Dukungan Hardware

   Base register (pasangan antara base dan limit register) hanya cukup untuk memodelkan single atau multiple partisi. Sementara untuk paging dan segmentasi diperlukan tabel pemetaan alamat.

2. Kinerja

   Waktu yang dibutuhkan untuk memetakan dari alamat logika ke alamat fisik akan bertambah, jika algoritma semakin kompleks. Untuk sistem yang sederhana, hanya dibutuhkan untuk membandingkan atau menambahkan alamat logika (operasinya cepat). Sedangkan pada paging dan segmentasi, proses akan menjadi cepat jika tabel diimplementasikan dengan menggunakan register yang cepat . Jika tabel ada di memori, maka kinerjanya akan berkurang .

3. Fragmentasi

   Sistem dengan multiprogramming akan lebih efisien jika menggunakan multiprogramming pada tingkat yang lebih tiggi. Jika terdapat sekeumpulan proses, tingkat multiprogramming dapat dinaikkan hanya dengan cara mengemas lebih banyak proses ke memori. Sehingga fragmentasi yang ada dalam memory harus dikurangi.

   Sistem dengan unit alokasi berukuran tertentu seperti single partisi atau paging akan mengalami masalah dengan adanya fragmentasi internal. Sementara itu sistem dengan partisi yang tidak sama akan mengalami masalah fragmentasi eksternal.

4. Relokasi

   Salah satu solusi untuk menanggulangi fragmentasi eksternal adalah kompaksi. Kompaksi mengijinkan suatu program digeser dimemori tanpa ada perubahan pada program itu sendiri. Hal ini berarti alamat logika dapat direlokasikan secara dinamis pada saat eksekusi. Jika alamat-alamat direlokasi hanya pada load time maka kompaksi tidak dapat dilakukan.

5. Swapping

   Semua algoritma manajemen memori pasti mengalami swapping. Interval terjadinya swapping diatur oleh system operasi melalui kebijakan CPU-schedulling.

6. Sharing

   Cara lain untuk menaikkan tingkat multiprogramming adalah dengan cara sharing kode dan data antara user-user yang berbeda. Umumnya proses sharing

menggunakan paging atau segmentasi untuk menunjukkan suatu paket informasi (page-page atau segment-segment) yang dapat dipakai bersama-sama. Sharing disini berarti menjalankan banyak proses dalam memori dengan jumlah yang terbatas.

7. Proteksi

   Jika digunakan paging atau segmentasi, seksi-seksi yang berbeda dari suatu user program dapat dideklarasikan dalam bentuk *execute-onl, read-only, atau read-write.* Batasan ini penting terutama kode atau data yang dipakai bersama oleh proses-proses.