INSTITUT TEKNOLOGI
TELKOM

Manajemen Memori

(Pertemuan ke-12)

Oktober 2010

Manajemen Memori

- Manajemen memori dilakukan dengan cara membagi-bagi memori untuk mengakomodasi banyak proses
- Manajemen memori dilakukan untuk menjamin agar setiap proses yang *ready* dapat segera memanfaatkan *processor time*

Requirement Manajemen Memori

- Terdapat 5 *requirement* manajemen memori:
  - *Relocation*
  - *Protection*
  - *Sharing*
  - *Logical organization*
  - *Physical organization*

Requirement Manajemen Memori:
Mengapa *Relocation* perlu ditangani ?

- *Programmer* tidak tahu di bagian memori yang mana program akan ditaruh pada saat dieksekusi
- Pada saat program dieksekusi, dimungkinkan program tersebut akan *di-swap* ke disk dan kemudian diambil lagi dari disk untuk ditaruh di memori dengan lokasi yang berbeda dengan lokasi sebelumnya (terjadi *relocation*)
- Diperlukan adanya translasi antara alamat program dengan alamat fisik memori

Teknik *Relocation (1)*

- Mengapa *relocation* penting dalam manajemen memori ?
  - Jika suatu program di-*load* ke memori, maka alamat lokasi memori (alamat absolut atau alamat fisik) yang akan ditempati harus ditentukan
  - Alamat absolut suatu program dapat berubah-ubah sebagai akibat:
    - *Swapping*
    - *Compaction*

Teknik *Relocation* (2)

- Jenis-jenis alamat data:
  - Alamat *Logical*
    - Alamat suatu *word* relatif terhadap titik referensi tertentu
    - Harus dilakukan translasi terhadap alamat fisik
  - Alamat Relatif
    - Alamat yang menunjukkan lokasi relatif terhadap alamat awal suatu program
    - Dapat terdiri dari:
      - Alamat instruksi dalam pencabangan
      - Alamat instruksi call
      - Alamat data
  - Alamat Fisik
    - Alamat mutlak/absolut
    - Merupakan alamat sebenarnya dari suatu memori

Teknik *Relocation (3)*

INSTITUT TEKNOLOGI
TELKOM

- Alamat relatif dan alamat absolut:

Symbolic addresses
PROGRAM
JUMP X
X
LOAD Y
DATA
Y

(a) Object module

Absolute addresses
1024
PROGRAM
JUMP 1424
1424
LOAD 2224
DATA
2224

(b) Absolute load module

Relative addresses
0
PROGRAM
JUMP 400
400
LOAD 1200
DATA
1200

(c) Relative load module

Teknik *Relocation* (4)

- Gambar mekanisme penempat-an program ke memori:

Relative address
Base Register
Adder
Absolute address
Process Control Block
Program
Data
Stack
Process image in main memory

Teknik *Relocation* (5)

- Jenis-jenis *register* yang digunakan:
  - *Base register*
    - Alamat awal suatu proses (tiap proses berbeda-beda)
    - Ditentukan oleh sistem operasi
  - *Bound register*
    - Alamat akhir suatu proses
    - *Bound register* = *base register* + panjang proses
  - Kedua alamat di-*set* pada saat proses di-*load* atau kembali dari *swap*
  - Setiap proses mempunyai nilai *base register* dan *bound register* sendiri-sendiri

Teknik *Relocation (6)*

- Mekanisme *relocation*:
  - Alamat *base register* ditambahkan dengan alamat relatif sehingga diperoleh alamat absolut

Requirement Manajemen Memori: Mengapa *Protection* perlu ditangani ?

- Suatu proses **tidak boleh** mengakses lokasi memori proses yang lain tanpa ijin
- Lokasi program di memori tidak tentu → alamat mutlak pada saat *compile* tidak dapat diketahui
- Penanganan proteksi memori diserahkan kepada prosesor (secara *hardware*), bukan bagian sistem operasi (*software*)
  – Sistem operasi tidak dapat memperkirakan penggunaan semua referensi memori dari setiap program

Teknik *Protection*

INSTITUT TEKNOLOGI
TELKOM

- Gambar mekanisme pemeriksa-an alamat relatif apakah valid atau tidak:

Relative address

Base Register

Adder

Absolute address

Bounds Register

Comparator

Interrupt to operating system

Process Control Block

Program

Data

Stack

Process image in main memory

Teknik *Relocation (6)*

- Mekanisme *relocation*:
  - Alamat *base register* ditambahkan dengan alamat relatif sehingga diperoleh alamat absolut
  - Alamat yang diperoleh dibandingkan dengan nilai-nilai pada *bound register*
  - Jika nilainya berada di dalam *bound* → instruksi dilanjutkan
  - Jika nilainya di luar *bound* → dihasilkan *interrupt*:
    - Sistem operasi harus mengambil tindakan lebih lanjut
    - Merupakan mekanisme proteksi terhadap pengaksesan data proses yang lain

Requirement Manajemen Memori:
Mengapa *Sharing* perlu ditangani ?

- Harus dimungkinkan suatu lokasi memori dapat diakses oleh lebih dari satu proses (secara legal)
- Beberapa proses yang memerlukan data yang sama, maka tidak perlu setiap proses meng-copy data dari disk ke memori, cukup sebuah copy saja

Requirement Manajemen Memori:

Mengapa *Logical Organization* perlu ditangani ?

- *Main memory* dikelompokkan secara linier atau berdimensi satu yang berupa ruang alamat yang terdiri dari deretan byte atau *word*
- Sebagian besar program dibuat dalam bentuk modul → penempatan di memori tidak linier → diperlukan pengaturan secara lojik
- Kelebihan program dalam bentuk modul:
  - Modul dapat ditulis dan di-*compile* secara terpisah
  - Setiap modul dapat diberi tingkatan proteksi berbeda-beda (*read-only, execute only*)
  - Module dapat di-*share* oleh beberapa proses

Requirement Manajemen Memori:

Mengapa *Physical Organization* perlu ditangani ?

- *Memory* dikelompokkan menjadi memori utama dan memori sekunder
- Diperlukan metode yang mengatur aliran data dari memori utama ke memori sekunder dan sebaliknya
- Pengaturan tersebut dilakukan oleh sistem operasi, bukan programmer, karena:
  - *Programmer* tidak tahu berapa ruang memori yang tersedia pada saat program dijalankan
  - Bila ruang memori yang tersedia tidak cukup untuk menampung program dan data → sistem dapat melakukan *overlay*
    - ***Overlay*** adalah menempatkan beberapa bagian modul/program pada area memori yang sama secara bergantian

Teknik Manajemen Memori

- Beberapa teknik manajemen memori yang ada:
  - Partisi
    - Partisi tetap (*fixed*)
      - Partisi berukuran sama
      - Ukuran partisi berbeda-beda
    - Partisi dinamis
  - *Paging* sederhana
  - Segmentasi sederhana
  - *Virtual-memory (akan dibahas secara khusus)*
    - *Virtual-memory paging*
    - *Virtual-memory segmentation*

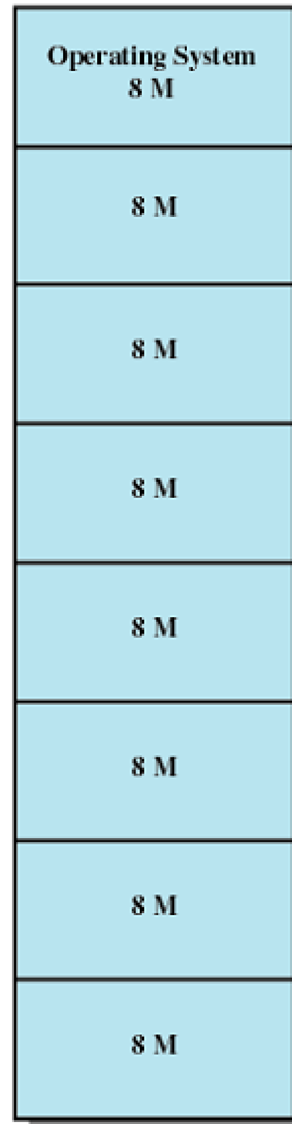
Partisi Memori Tetap *(fixed)* (1)

- Sebelum digunakan memori terlebih dahulu dipartisi (ukuran partisi tidak berubah)
- Model ukuran partisi:
  - Partisi berukuran sama:
    - Setiap proses yang ukurannya lebih kecil atau sama dengan ukuran partisi dapat menempati partisi tersebut
    - Jika semua partisi telah terisi, maka sistem operasi akan melakukan *swap* terhadap proses yang sudah tidak aktif
    - Dimungkinkan adanya program yang ukurannya lebih besar daripada ukuran partisi yang tersedia → *programmer* harus merancang program dengan *overlay*
    - Penggunaan memori sangat tidak efisien
      - Misal bila ukuran partisi adalah 8 MB, maka program berukuran 2 MB akan menyisakan ruang memori sebesar 6 MB
  - Partisi berukuran tidak sama:
    - Lebih baik daripada partisi berukuran sama:
      - Penggunaan memori lebih efisien
      - Tidak perlu *overlay*

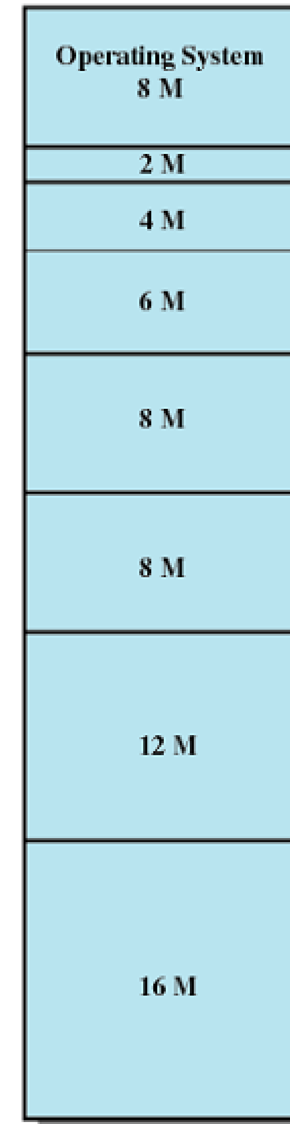
Partisi Memori Tetap *(fixed)* (2)

Jenis partisi tetap:

- *Equal-size*
- *Unequal-size*



(a) Equal-size partitions



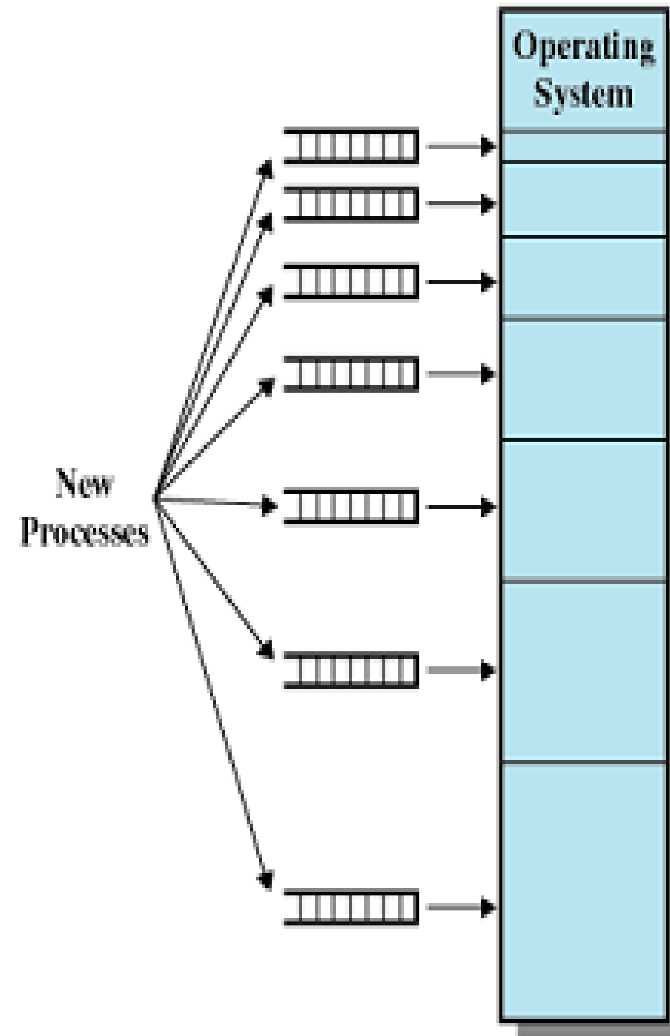
(b) Unequal-size partitions

Partisi Memori Tetap *(fixed)* (3)

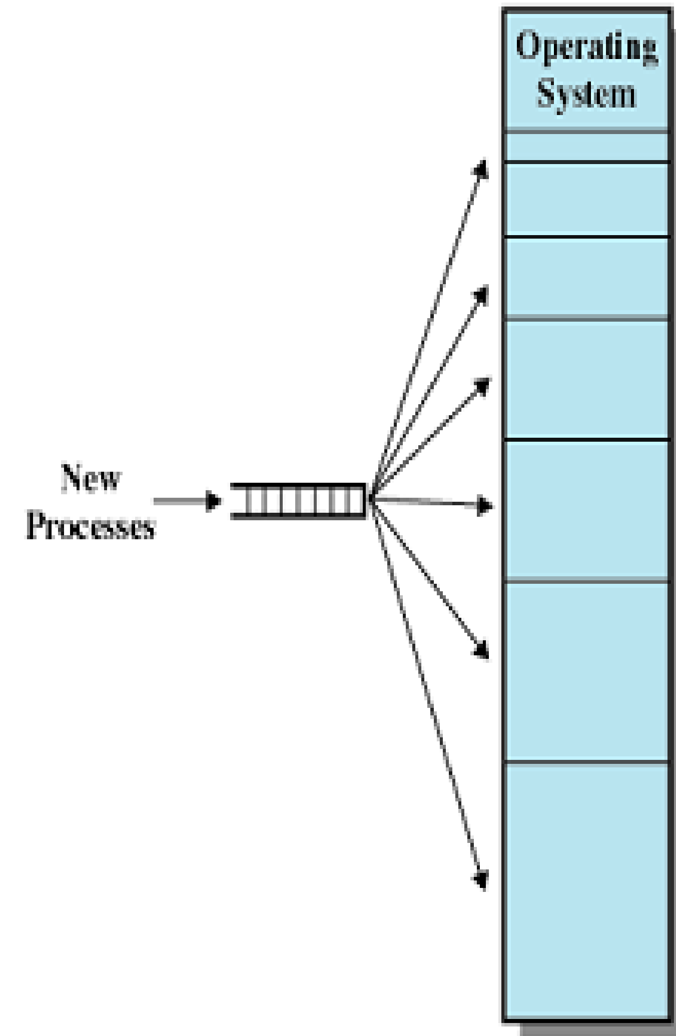
- Algoritma penempatan (*placement*)
  - Partisi berukuran sama
    - Algoritmanya sederhana, partisi yang mana saja asalkan kosong boleh ditempati, karena ukurannya sama
  - Partisi berukuran berbeda
    - Setiap proses ditempatkan pada partisi yang menyisakan ruang bebas terkecil
    - Terdapat 2 model antrian:
      - Satu antrian – satu partisi
      - Satu antrian – banyak partisi

Partisi Memori Tetap *(fixed)* (4)

- Model antrian:



(a) One process queue per partition



(b) Single queue

Partisi Memori Tetap *(fixed)* (5)

- Satu partisi - satu antrian
  - Setiap proses antri pada partisi yang berukuran sedikit lebih besar atau sama dengan ukuran proses
  - Kelebihan:
    - + Dapat meminimalisir ruang memori yang terbuang
  - Kekurangan:
    - Diperlukan penjadualan antrian
    - Ada kemungkinan efisiensi secara keseluruhan tidak optimal
      - Misal pada model partisi di slide hal 10 tidak ada proses yang berukuran di antara 12 – 16 MB → partisi 16 MB tidak akan pernah digunakan
- Banyak partisi - satu antrian
  - Setiap proses dapat menempati di sembarang partisi yang sedang tidak digunakan
  - Dipilih partisi yang menyisakan ruang memori terkecil
  - Bila seluruh partisi telah diisi → dilakukan *swapping*

Partisi Memori Tetap *(fixed)* (6)

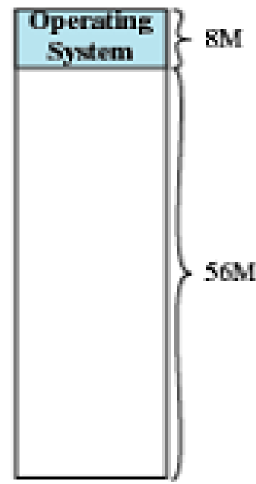
- Kelebihan:
  - + Mudah diimplementasikan
  - + *Overhead* sistem operasi hanya sedikit
- Kekurangan:
  - – Tidak efisien dalam penggunaan memori akibat terjadi fragmentasi internal
    - **Fragmentasi internal**: sisa ruang memori yang terjadi jika ukuran proses lebih kecil daripada partisi yang digunakan
  - – Jumlah maksimum proses yang aktif adalah tetap dan terbatas (bergantung jumlah partisi)

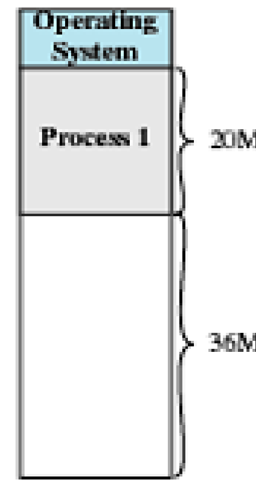
Partisi Memori Dinamis (1)

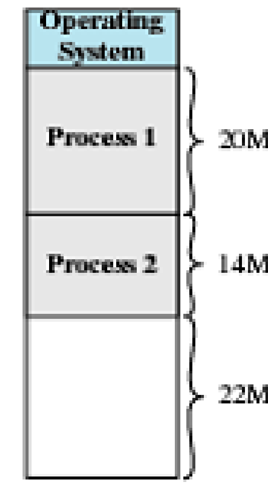
- Jumlah dan ukuran partisi tidak tetap (variabel)
- Ukuran partisi sama dengan ukuran proses yang akan menempatinya untuk pertama kali atau sesudah pemadatan (*compaction*) → tidak terjadi fragmentasi internal
- Dapat terjadi fragmentasi eksternal
  - **Fragmentasi eksternal**: sisa ruang memori yang terjadi jika ukuran proses lebih kecil daripada ruang memori yang disediakan (dibebaskan)
  - **Solusi**: dilakukan *compaction* sehingga sisa-sisa ruang memori terkumpul menjadi satu → sisa ruang memori menjadi besar

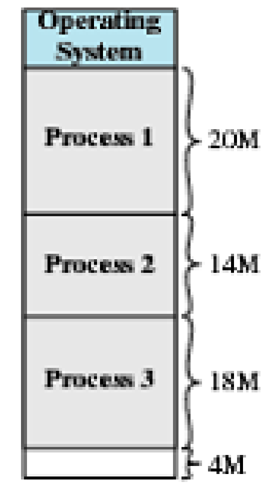
Partisi Memori Dinamis (2)

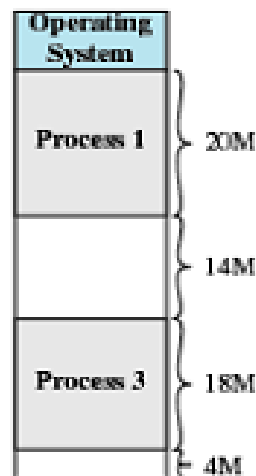
- Contoh partisi dinamis

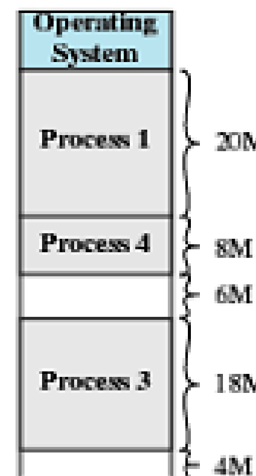


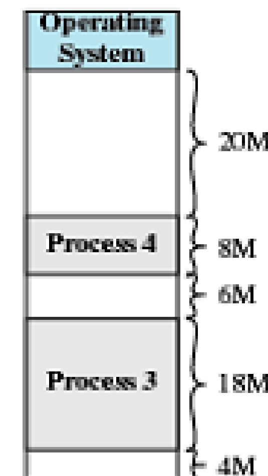


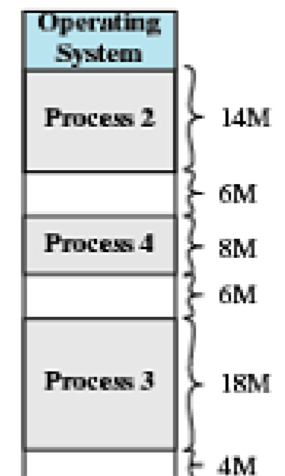












Partisi Memori Dinamis (3)

Keterangan:

a. Tersedia 64 MB memori, 8 MB untuk sistem operasi

b. Proses 1 aktif dan membutuhkan 20 MB

c. Proses 2 aktif dan memerlukan 14 MB

d. Proses 3 aktif dan memerlukan 18 MB, sisa memori tinggal 4 MB

e. Proses 2 selesai

f. Proses 4 aktif dan memerlukan 8 MB → ditempatkan di ruang memori bekas proses 2 → terjadi fragmentasi eksternal sebesar 6 MB

g. Proses 1 telah selesai → tersedia ruang bebas sebesar 20 MB

h. Proses 2 aktif lagi dan titempatkan pada lokasi bekas proses 1 → terjadi fragmentasi eksternal sebesar 6 MB

Partisi Memori Dinamis (4)

- Algoritma penempatan (*placement*)
  - Permasalahan:
    - Kapan saat untuk melakukan *loading* dan *swapping* ?
    - Bila terdapat lebih dari satu blok memori yang dapat dibebaskan, blok memori yang mana yang akan di-*swap* ?
  - Algoritma yang dapat digunakan:
    - *Best-fit*
    - *First-fit*
    - *Next-fit*

Partisi Memori Dinamis (5)

- Algoritma *Best-fit*:
  - Memilih blok memori yang paling sedikit menyisakan ruang memori
  - Biasanya performansi secara keseluruhan adalah yang paling jelek:
    - Proses pencarian lebih lama dan membebani prosesor
    - Sisa memori berukuran kecil-kecil lebih cepat terbentuk → *Compaction* harus lebih sering dilakukan daripada algoritma yang lain

Partisi Memori Dinamis (6)

- Algoritma *First-fit*:
  - Pencarian blok memori kosong dimulai dari awal
  - Blok memori yang dipilih adalah blok memori yang pertama kali ditemukan dan ukurannya sesuai
  - Merupakan algoritma yang paling baik:
    - Paling cepat
    - Paling sederhana
  - Pencarian akan melewati sejumlah proses yang terletak pada bagian ujung awal memori sebelum menemukan blok memori yang bebas

Partisi Memori Dinamis (7)

- Algoritma *Next-fit*:
  - Pencarian blok memori kosong dimulai dari lokasi *placement* terakhir
  - Lebih jelek dibanding *First-fit*, karena:
    - Blok memori yang ditemukan sering berada pada ujung akhir memori yang merupakan blok memori berukuran paling besar:
      - Blok memori yang besar akan lebih cepat terpartisi menjadi blok memori yang lebih kecil
      - *Compaction* untuk memperoleh blok memori berukuran besar pada ujung akhir memori harus lebih sering dilakukan daripada *First-fit*

Partisi Memori Dinamis (8)

INSTITUT TEKNOLOGI
TELKOM

- Contoh algoritma *placement*
- Ukuran proses baru = 16 MB
- Memori sisa:
  - Best-fit = 2 MB
  - First-fit = 6 MB
  - Next-fit = 20 MB

Last allocated block (14K)

8M
12M
22M
18M
8M
6M
14M
36M

(a) Before

First Fit
Best Fit
Next Fit

8M
12M
6M
2M
8M
6M
14M
20 M

(b) After

Allocated block
Free block
Possible new allocation

Partisi Memori Dinamis (9)

- Kelebihan:
  + Tidak terjadi fragmentasi internal
  + Penggunaan memori lebih efisien
  + Jumlah proses aktif lebih fleksibel (tidak tetap)
- Kekurangan:
  - Implementasinya lebih susah
  - Dapat terjadi fragmentasi eksternal
  - Terjadi overhead penggunaan prosesor:
    - Untuk *compaction*
    - Untuk menjalankan algoritma

Pustaka

[STA09] Stallings, William. 2009. *Operating System: Internal and Design Principles*. 6th edition. Prentice Hall