Optimasi Algoritma Banker dengan Algoritma Greedy

Daniel Prihartoni – 1359088
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13509088@std.stei.itb.ac.id

***Abstrak*— Pada konsep sistem operasi, dikenal adanya istilah *deadlock*. *Deadlock* terjadi pada sejumlah proses ketika masing-masing proses saling menunggu suatu kejadian yang hanya dapat terjadi jika disebabkan oleh proses lainnya. Salah satu strategi untuk mengatasi *deadlock* adalah dengan melakukan penjadwalan penggunaan *resource* oleh proses. Algoritma penjadwalan ini dikenal dengan nama Algoritma Banker yang akan dioptimasi dengan menggunakan algoritma Greedy.**

Kata kunci—Sistem operasi, deadlock, algoritma Banker, algoritma Greedy .

I. PENDAHULUAN

Salah satu peran sistem operasi adalah berkaitan dengan manajemen proses dan *resource*. Ketika komputer dijalankan, sejumlah proses akan dilakukan oleh CPU dengan menggunakan *resource* yang ada sesuai dengan kebutuhan. Suatu *resource* dapat berupa sebuah perangkat keras (contoh : *printer*) yang terhubung dengan komputer ataupun sepenggal informasi yang tersimpan pada media penyimpanan. Peran sistem operasi adalah untuk mengalokasikan satu atau lebih *resource* untuk digunakan oleh proses tertentu.

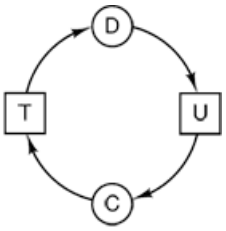
Pada praktiknya, sering kali sebuah proses membutuhkan suatu *resource* yang juga dibutuhkan oleh proses lain. Hal ini biasa dikenal dengan nama *resource sharing*. Jika penjadwalan berjalan dengan baik, maka setiap proses akan mendapatkan *resource* yang dibutuhkan sehingga proses-proses tersebut dapat selesai.

Secara formal, *deadlock* didefinisikan sebagai kejadian yang terjadi pada sejumlah proses ketika masing-masing proses menunggu suatu kejadian yang disebabkan oleh proses lain [1]. Masing-masing proses tidak akan menyebabkan kejadian yang diharapkan proses lain karena proses tersebut juga menunggu sebuah kejadian dari proses lain. Hal ini menyebabkan masing-masing proses akan saling menunggu selamanya.

Yang dimaksud dengan kejadian pada makalah ini adalah kondisi ketika sebuah proses melepaskan satu atau lebih *resource*. Dengan kata lain definisi sebelumnya menjadi masing-masing proses akan menunggu sebuah *resource* yang sedang digunakan proses yang juga sedang menunggu. *Deadlock* semacam ini dikenal dengan nama *resource deadlock*.

Ada empat kondisi yang harus dipenuhi ketika sejumlah proses dikatakan dalam keadaan *resource deadlock* [2], yaitu :

1. *Mutual exclusion*, setiap *resource* sedang digunakan oleh tepat satu proses.
2. *Hold and wait*, proses yang sedang menggunakan *resource* meminta hak akses untuk *resource* lain.
3. *No preemption,* setiap *resource* tidak dapat diambil secara paksa.
4. *Circular wait*, terdapat lintasan sirkuler dari dua atau lebih proses yang sedang menunggu hak akses penggunaan dari proses lain yang ada di lintasan sirkuler.



Gambar 1 – Kondisi *deadlock*

Ada empat strategi untuk menangani *deadlock* yaitu :

1. Mengabaikan kejadian *deadlock*, yaitu dengan prediksi bahwa *deadlock* hannya terjadi sekali dalam lima tahun
2. *Detection and recovery*, yaitu apabila terjadi *deadlock*, lakukan sesuatu.
3. *Dynamic avoidance*, yaitu dengan melakukan perhitungan matang sebelum mengalokasi *resource*.
4. Mencegah, yaitu dengan melanggar salah satu dari empat kodisi terjadinya *deadlock*.

II. DASAR TEORI

Pada makalah ini hanya akan dibahas mengenai strategi yang ketiga yaitu *dynamic avoidance*. Pada *dynamic avoidance* dikenal istilah stata aman dan stata tidak aman. Suatu kondisi dikatakan aman jika urutan penjadwalan penggunaan *resource* menjamin setiap proses untuk dapat diselesaikan. Contohnya pada gambar 2, jika proses B meminta 2 resource, sedangkan *resource* yang tersisa adalah 3, maka proses B akan selesai dan mengembalikan *resource* yang telah dipakai yaitu sebanyak 4. Dengan melakukan hal yang sama selanjutnya pada proses A dan

C, maka keseluruhan proses dapat diselesaikan

	Has	Max
A	3	9
B	2	4
C	2	7

Free: 3

Gambar 2 – Contoh kondisi aman

Sementara itu kondisi yang dikatakan tidak aman adalah jika setiap proses tidak dapat dijamin dapat diselesaikan tanpa menghadapi kondisi *deadlock*. Kondisi tidak aman bukan berarti sedang mengalami *deadlock*, tetapi berpotensi untuk terjadi *deadlock*. Contohnya pada gambar 3, ketika proses B diseleasikan, *resource* yang dikembalikan tidak cukup untuk menyelesaikan proses A maupun proses C.

	Has	Max
A	4	9
B	2	4
C	2	7

Free: 2

Gambar 3 – Contoh kondisi tidak aman

Untuk menghindari terjadinya kondisi *deadlock* digunakan algoritma penjadwalan yang dikenal dengan nama algoritma Banker. Algoritma ini memodelkan seorang bankir yang melayani peminjaman uang untuk beberapa orang di suatu kota kecil. Bankir akan memeriksa agar setiap pemberian pinjaman uang tidak menyebabkannya ke kondisi tidak aman. Bankir tidak menyediakan semua pinjaman yang dibutuhkan sekaligus tetapi secara bertahap. Dengan asumsi si peminjam akan mengembalikan pinjamannya ketika urusannya sudah selesai, maka pinjaman yang lain dapat dipenuhi.

Meskipun pada penjelasan sebelumnya selalu diasumsikan bahwa masing-masing proses menggunakan satu *resource* namun pada kenyataannya yang sering terjadi adalah sejumlah proses berbagi sejumlah *resource* [3]. Alokasi *resource* untuk setiap proses dimodelkan dengan matriks seperti pada gambar 3.

Process	Tape drives	Plotters	Scanners	CD ROMs
A	3	0	1	1
B	0	1	0	0
C	1	1	1	0
D	1	1	0	1
E	0	0	0	0

Resources assigned

Process	Tape drives	Plotters	Scanners	CD ROMs
A	1	1	0	0
B	0	1	1	2
C	3	1	0	0
D	0	0	1	0
E	2	1	1	0

Resources still needed

E = (6342)
P = (5322)
A = (1020)

Gambar 4 – Model alokasi untuk sejumlah *resource*

Untuk setiap kondisi dibutuhkan dua buah matriks yang menunjukkan jumlah *resourcre* yang sudah dialokasikan (gambar 4, matriks sebelah kiri) dan *resource* yang masih dibutuhkan oleh sejumlah proses (gambar 4, matriks sebelah kanan). Adapun keterangan lainnya yang dibutuhkan adalah :

1. *E*, yaitu jumlah *resource* yang tersedia pada awalnya.
2. *P*, yaitu jumlah *resource* yang dialokasikan untuk seluruh proses yang ada.
3. *A*, yaitu jumlah *resource* yang tersisa setelah dialokasikan.

Adapun penerapan algoritma Banker yang digunakan adalah sebagai berikut :

1. Cari satu baris, misal *R*, pada matriks *resource* yang masih dibutuhkan dengan kriteria setiap nilai berurutannya lebih kecil atau sama dengan *A*. Jika tidak ada yang memenuhi, maka sistem dalam keadaan *deadlock*.
2. Asumsikan proses menggunakan semua *resource* yang dibutuhkan sekaligus pada satu waktu.
3. Ulangi langkah 1 dan langkah 2 hingga semua proses selesai.

Pada langkah pertama dari algoritma Banker, sangat dimungkinkan terjadi beberapa kandidat proses yang memungkinkan untuk dimasukkan ke tahap selanjutnya. Makalah ini akan membahas mengenai optimasi pilihan yang diambil dengan memanfaatkan algoritma Greedy.

III. Analisis Pemecahan Masalah

Algoritma Greedy adalah metode untuk memecahkan persoalan yang berhubungan dengan optimasi. Algoritma ini sangat populer karena sederhana dan lempang (straightforward). Dari asal katanya *greedy* berarti rakus atau tamak. Prinsip utama *greedy* adalah mengambil langkah yang paling baik yang dapat diambil pada suatu saat tertentu.

Algoritma Greedy akan menghasilkan solusi langkah per langkah yang paling optimum dari sekumpulan kemungkinan solusi. Pendekatan yang digunakan pada algoritma Greedy adalah dengan membentuk pilihan terbaik untuk setiap langkah (optimum lokal) dengan harapan selanjutnya menghasilkan langkah-langkah yang paling optimum (optimum global).

Persoalan optimasi dengan menggunakan algoritma Greedy dimodelkan dengan beberapa elemen [4] yaitu :

1. Himpunan kandidat (C), yaitu berisi elemen - elemen pembentuk solusi.
2. Himpunan solusi (S), yaitu berisi kandidat - kandidat yang terpilih sebagai solusi persoalan.
3. Fungsi seleksi, yaitu fungsi untuk memilih kandidat yang memungkinkan mencapai solusi.
4. Fungsi kelayakan, yaitu fungsi untuk memeriksa kandidat agar tidak melanggar kendala.
5. Fungsi obyektif, yaitu fungsi yang mengoptimumkan pilihan yang ada.

Untuk dapat menggunakan algoritma Greedy, pertama permasalahan terkait alokasi *resource* untuk masing-masing proses ini akan dimodelkan pada elemen-elemen yang telah disebutkan sebelumnya :

1. Himpunan kandidat adalah semua proses yang masih menunggu untuk mendapatkan akses *resource*.
2. Himpunan solusi adalah satu proses yang dipilih untuk setiap kali giliran pengalokasian *resource*.
3. Fungsi seleksi adalah fungsi yang memilih nilai paling besar untuk setiap kemungkinan.
4. Fungsi kelayakan adalah fungsi yang menentukan kriteria layak untuk sebuah proses.
5. Fungsi obyektif adalah fungsi yang memberikan penilaian terhadap setiap proses yang mungkin untuk diselesaikan

Strategi Greedy yang digunakan adalah dengan selalu memilih proses yang telah dan akan menggunakan *resource* paling banyak yang masih mungkin untuk dipilih. Dengan memilih proses yang total *resource*-nya paling banyak, proses tersebut akan mengembalikan *resource* yang paling banyak pula sehingga diharapkan untuk memenuhi kebutuhan proses akan *resource* selanjutnya hingga selesai.

Di bawah ini adalah *pseudocode* hasil pemodelan masalah alokasi *resource* dengan algoritma Banker dengan menggunakan algoritma Greedy.

```
function Layak(proc : array of process, avail_res : array of process) → boolean
{ Fungsi ini adalah fungsi kelayakan dari model algoritma Greedy. Fungsi ini menentukan layak atau tidaknya sebuah proses untuk dikerjakan dengan batasan jumlah resource yang dibutuhkan suatu proses yang akan dikerjakan tidak melebihi jumlah resource yang tersedia.}

  value : boolean
  i     : integer

  value ← true
  i ← 1
  while (value and (i <= sizeof(proc))) do
    if (proc[i] > avail_res[i]) then
      value <-- false

    i <-- i + 1

→ value
```

```
function Obyektif(assgned : array of process, still_needed : array of process) → integer
{ Fungsi ini adalah fungsi obyektif dari model algoritma Greedy. Fungsi ini memberi nilai terhadap setiap proses berdasarkan strategi Greedy yang digunakan yaitu dengan menjumlahkan resource yang sudah dialokasi dengan resource yang akan dialokasi. }

  i     : integer
  value : integer

  value ← 0
  for i ← 1 to sizeof(assgned) do
    value ← assgned[i] + still_needed[i]

→ value
```

```
function Seleksi(assgned : array of array of process, still_needed : array of array of process, avail_res : array of process) --> array of process
{ Fungsi ini adalah fungsi seleksi pada dari model algoritma Greedy. Fungsi ini akan memilih satu dari sejumlah kemungkinan proses yang mungkin untuk dikerjakan berdasarkan strategi Greedy yang digunakan yaitu dengan memilih proses yang memiliki nilai Obyektif paling besar. }

  i, temp, value : integer

  value ← 1
  temp ← Obyektif(assgned[1],still_needed[1])
  for i ← 1 to sizeof (assgned) do
    if (Layak(assgned[i], avail_res)) then
      if (Obyektif(assgned[i],still_needed[i]) > temp) then
        value ← i
        temp ← Obyektif(assgned[i],still_needed[i])

→ value
```

```
function CountAvailRes (existingres : array of process, assgned : array of array of process) → array of process
{ Fungsi ini digunakan untuk menghitung jumlah resource yang masih tersedia setelah dilakukan alokasi pertama kali terhadap proses yang ada. }

  usedres      : array of process
  i,j, temp    : integer

  for i ← 1 to sizeof(existingres) do
    temp ← 0
    for j ← 1 to sizeof(assgned) do
      temp ← temp + assgned[j][i]

    usedres[i] ← temp

  for i ← 1 to sizeof(usedres) do
    usedres[i] ← existingres[i] - usedres[i]

→ usedres
```

```
procedure AssignRes(turn : integer, assgned : array of array of process, still_needed : array of array of process, avail_res : array of process)
{ Prosedur ini digunakan untuk melakukan alokasi resource terhadap proses dari hasil fungsi seleksi. Prosedur ini juga akan memperbaharui nilai resource yang tersedia setelah dilakukan alokasi dan nilai resource yang dibutuhkan oleh suatu proses }
```

```
    i       : integer

    for i ←1 to sizeof(assgned) do
        assgned[turn][i] ← assgned[turn][i] +
still_needed[turn][i]
        avail_res[i] <-- avail_res[i] - still_needed[turn][i]
        still_needed[turn][i] ← 0

→ turn
```

```
procedure TakeResBack(idx : integer, assgned : array of
array of process, avail_res : array of process)
{ Prosedur ini digunakan untuk mengembalikan resource
yang diasumsikan sudah selesai digunakan oleh suatu
proses. }

    i       : integer

    for i ← 1 to sizeof(avail_res) do
        avail_res[i] ← avail_res[i] + assgned[idx][i]
        assgned[idx][i] ← 0
```

```
function isFinish(proc : array of array of process) →
boolean
{ Fungsi ini menentukan keberadaan proses yang masih
membutuhkan resource. }

    i,j,c    : integer

    i ← 1
    c ← 0
    for each (proc) do
        j ← 1
        for each (proc[i]) do
            if (proc[i][j] = 0) then
                c ← c + 1
            j ← j+1
        i ← i + 1

    if (c = sizeof(proc[1]) * sizeof(proc[1][1])) then
        → true
    else
        → false
```

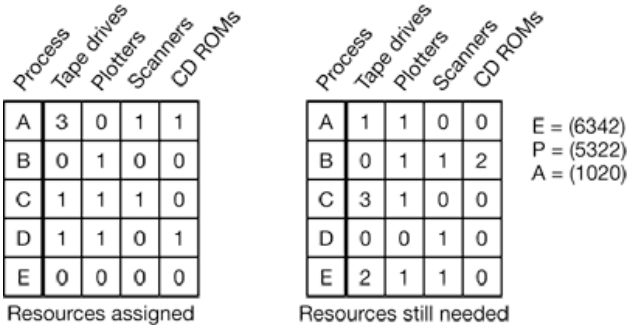
```
{PROGRAM UTAMA}

    E, A            : array of process
    assgned         : array of array of process
    still_needed    : array of array of process
    turn            : integer
    finish          : boolean


    { isi E dengan nilai pre-defined }
    { isi assgned dengan nilai pre-defined }
    { isi still_needed dengan nilai pre-defined }
    finish ← false
    A ← CountAvailRes(E, assgned)

    while not finish do
        turn ← Seleksi (assgned, still_needed, A)
        AssignRes(assgned, still_needed, A)
        TakeResBack(turn, assgned, A)
        if (isFinish(still_needed)) then
            finish ← true
```

Sebagai prakondisi, diasumsikan bahwa masalah berupa tabel proses dan jumlah *resource* yang dimilki sudah terdefinisi. Pada saat dijalankan, program akan melakukan iterasi hingga seluruh proses selesai. Untuk setiap iterasi, program akan memilih satu proses yang memenuhi kriteria Greedy yang telah ditetapkan sebelumnya, lalu melakukan alokasi *resource* dan mengembalikan *resource* tersebut sebagai *resource* yang dapat digunakan lagi.

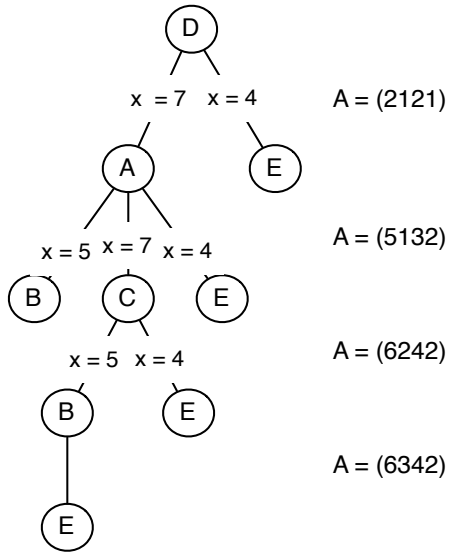
IV. HASIL PENGUJIAN

Untuk contoh kasus akan digunakan kasus seperti pada gambar 5. Penggunaan algoritma Greedy tidak berpengaruh karena pada langkah pertama, hanya terdapat satu proses yang mungkin diselesaikan yaitu proses D. Pada langkah selanjutnya nilai *A* akan berubah menjadi (2121) dan terdapat 2 pilihan proses untuk diseleksaikan yaitu proses A dan proses E. Algoritma Greedy akan memilih proses A untuk diselesaikan terlebih dulu karena proses A memiliki nilai 7 sementara proses E memiliki nilai 4. Pada langkah ketiga, nilai *A* berubah menjadi (5132) dan seterusnya hingga semua proses selesai.

Dari analisis terhadap setiap kemungkinan pilihan yang muncul untuk setiap langkah pada kondisi seperti gambar 4, algoritma Greedy baru akan bekerja ketikia pilihan proses yang ada lebih dari satu. Pada gambar 6 ditunjukkan pohon yang menunjukkan langkah pengalokasian *resource* untuk setiap proses.



Gambar 5 – Contoh kasus untuk menguji algoritma Greedy



Gambar 6 – Pohon penelusuran langkah untuk kasus pada gambar 5

Pada gambar 6, didapat bahwa algoritma Greedy akan bekerja pada *node* level 2, 3, dan 4. Untuk setiap langkah dengan pilihan lebih dari satu, algoritma Greedy akan memilih proses dengan nilai *x* yang terbesar.

V. KESIMPULAN

Algoritma Banker pada dasarnya akan menangani alokasi *resource* dengan mangkus. Dapat dikatakan bahwa tanpa menggunakan algoritma Greedy sekalipun tidak akan menyebabkan *deadlock* pada sistem. Adapun penggunaan algoritma Greedy di sini hanya bertujuan untuk mengoptimasi pengalokasian *resource* agar selalu digunakan sebanyak mungkin pada kesempatan pertama. Pada kenyataannya, permintaan proses untuk mendapat *resource* tidak akan pernah statis seperti pada contoh. Jika melihat pada contoh sebuah komputer yang digunakan untuk berbagai keperluan seperti membuat dokumen, memutar musik, dan melakukan kompresi terhadap berkas secara bersamaan, suatu proses akan selalu datang bergantian. Dengan strategi Greedy, sistem operasi akan selalu mempunyai *resource* dengan nilai optimal yang mungkin dimiliki, sehingga ketika proses baru dengan kebutuhan *resource* yang tinggi masuk ke daftar proses, sistem dapat mengerjakanya langsung tanpa menunggu proses lain selesai dan mengembalikan *resource* hingga cukup.

DAFTAR REFERENSI

[1] Tanenbaum, A., *Modern Operating System*, 3rd ed, New Jersey : Prentice-Hall, 2009, bab 6, hal. 431 – 464.

[2] Coffman, E.G., Elphick, M.J., dan Shoshani, A., "System Deadlocks," *Computing Surveys*, vol. 3, hal. 67 -78, Juni 1971.

[3] Holt, R. C., "Some Deadlok Properties of Computer Systems," *Computing Surveys*, vol. 4, hal, 179 – 196, September 1972.

[4] Munir, R., *Diktat Kuliah IF3051 Strategi Algoritma*, Bandung, 2009.

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 9 Desember 2011

ttd

Daniel Prihartoni
13509088