Implementasi Algoritma Greedy dalam Alokasi Memori Kontigu pada Sistem Operasi

Kevin Roni - 13520114
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail (gmail): 13520114@std.stei.itb.ac.id

***Abstract*—Sistem operasi adalah program yang menjadi perantara antara pengguna komputer dengan perangkat keras komputer. Salah satu komponen penting dalam pembuatan sistem operasi adalah manajemen memori. Salah satu cara untuk melakukan manajemen memori adalah melakukan alokasi memori kontigu. Dalam dunia informatika, salah satu strategi untuk menyelesaikan masalah adalah dengan menggunakan strategi greedy. Dalam laporan ini, akan dibahas mengenai implementasi alokasi memori kontigu dengan menggunakan algoritma greedy.**

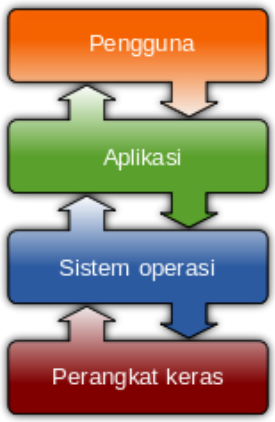
Keywords—Alokasi Memori Kontigu, Sistem Operasi, Algoritma Greedy

I. PENDAHULUAN

Salah satu komponen yang harus ada agar setiap komputer dapat berjalan dengan baik adalah sistem operasi. Sistem operasi sendiri adalah perangkat lunak sistem yang bertugas sebagai perantara antara pengguna komputer dengan perangkat keras komputer. Sistem operasi memiliki tugas untuk mengatur sumber daya dari perangkat keras komputer. Tanpa adanya sistem operasi pengguna tidak akan bisa mengutilasi program/aplikasi yang ada pada komputer, kecuali untuk program booting. Pada umumnya, ketika booting komputer, sistem operasi akan menjadi program pertama yang diletakkan pada memori komputer. Biasanya perangkat lunak lainnya akan dijalankan setelah sistem operasi berhasil dieksekusi. Sistem operasi kemudian akan bertindak sebagai pelayan untuk mengerjakan layanan inti perangkat lunak lainnya.

Sistem operasi memiliki banyak fitur. Seiring dengan berkembangnya teknologi, fitur-fitur yang ada di dalam sistem operasi juga semakin bertambah. Beberapa fitur umum yang sering terdapat di dalam sistem operasi adalah manajemen proses, file system, input/output, keamanan dan proteksi, jaringan, serta manajemen memori. Manajemen memori sendiri dapat diartikan sebagai proses untuk memasukkan program ke dalam memori. Manajemen memori bertugas untuk mengatur antrian program yang masuk ke dalam memori agar dapat dieksekusi. Salah satu cara program dapat mengaturnya adalah dengan melakukan alokasi memori kontigu. Alokasi memori dapat dilakukan dengan menggunakan metode first fit, best fit, dan worst fit.

Selama satu semester mempelajari mata kuliah Strategi Algoritma, penulis menyadari bahwa materi yang dipelajari pada kuliah sangat banyak aplikasinya dalam dunia nyata. Salah satunya adalah algoritma greedy. Algoritma greedy memiliki prinsip untuk mengambil pilihan yang terbaik tanpa melihat konsekuensi ke depannya dengan harapan dengan memilih optimum lokal pada setiap langkah akan menghasilkan solusi optimum global. Pada kesempatan kali ini, penulis akan menyelesaikan permasalahan alokasi memori kontigu pada sistem operasi dengan menggunakan algoritma Greedy.



Gambar 1. Grafik Struktur Sistem Komputer

Sumber: Golftheman

II. LANDASAN TEORI

Pada bab ini, akan dijelaskan teori-teori yang diperlukan dalam penyelesaian permasalahan alokasi memori kontigu, terdiri atas algoritma greedy, sistem operasi, manajemen memori dan alokasi memori kontigu

A. Algoritma Greedy

Algoritma greedy adalah salah satu dari sekian banyak algoritma yang dapat digunakan untuk menyelesaikan permasalahan. Pada dasarnya, algoritma greedy biasa digunakan untuk memecahkan permasalahan yang berhubungan dengan optimasi. Permasalahan optimasi memiliki dua jenis, yaitu maksimasi atau minimalisasi.

Salah satu contoh permasalahan optimasi yang paling populer adalah permasalahan penukaran uang. Misalkan ada uang senilai A. Kemudiakan tersedia beberapa koin dengan pecahan-pecahan tertentu. Berapa jumlah minimum koin yang diperlukan untuk penukaran tersebut. Salah satu metode paling sederhana untuk menyelesaikannya adalah dengan mengecek semua kemungkinan komposisi koin yang mungkin untuk menyusun pecahan uang tersebut kemudian mencari solusi dengan jumlah uang koin paling sedikit. Solusi ini sederhana namun memakan waktu yang sangat lama.

Algoritma greedy dapat didefinisikan sebagai algoritma untuk menyelesaikan masalah dengan langkah-langkah sehingga setiap langkahnya memiliki prinsip untuk mengambil opsi terbaik tanpa memikirkan konsekuensi ke depannya dengan harapan bahwa dengan memilih optimum lokal dapat menghasilkan optimum global.

Jika permasalahan penukaran uang ditinjau dengan pendekatan greedy, maka cara untuk menyelesaikannya adalah pada setiap langkah pemilihan koin, dicari koin dengan nilai terbesar dari koin-koin yang mungkin. Jika ditinjau kembali, berbeda dengan strategi brute force yang mengharuskan kita untuk mengecek semua kemungkinan, algoritma greedy tidak mengharuskan kita untuk mengecek semua kemungkinan.

Dalam penggunaannya, algoritma greedy memiliki elemen-elemen sebagai berikut:

1. Himpunan kandidiat (c)

   Berisikan elemen-elemen yang akan membentuk solusi

2. Himpunan solusi (s)

   Berisikan elemen-elemen yang sudah dipilih

3. Fungsi solusi

   Fungsi untuk menentukan kandidat yang terpilih sudah menghasilkan solusi

4. Fungsi seleksi

   Fungsi untuk memilih kandidat terbaik untuk solusi

5. Fungsi kelayakan

   Fungsi pembatas untuk memeriksa kelayakan setiap elemen pada himpunan kandidat

6. Fungsi obyektif

   Fungsi yang merupakan batasan optimum dari permasalahan yang akan diselesaikan

Secara umum, cara kerja algoritma greedy adalah sebagai berikut. Pertama-tama, semua elemen yang ada pada himpunan kandidat akan diseleksi oleh fungsi seleksi. Elemen-elemen terpilih akan diuji kelayakannya dengan fungsi kelayakan dan dikeluarkan dari himpunan kandidat. Jika elemen tersebut layak, maka elemen akan dimasukkan ke dalam himpunan solusi. Himpunan solusi akan diperiksa dengan fungsi solusi apakah sudah menghasilkan solusi. Kemudian himpunan solusi ini akan dioptimasi oleh fungsi obyektif. Secara umum, algoritma greedy memiliki skema sebagai berikut:

```
function greedy(C : himpunan_kandidat) → himpunan_solusi
{ Mengembalikan solusi dari persoalan optimasi dengan algoritma greedy }
Deklarasi
   x : kandidat
   S : himpunan_solusi

Algoritma:
   S ← {}      { inisialisasi S dengan kosong }
   while (not SOLUSI(S)) and (C ≠ {} ) do
      x ← SELEKSI(C)     { pilih sebuah kandidat dari C}
      C ← C – {x}        { buang x dari C karena sudah dipilih }
      if LAYAK(S ∪ {x}) then      { x memenuhi kelayakan untuk dimasukkan ke dalam himpunan solusi }
         S ← S ∪ {x}      { masukkan x ke dalam himpunan solusi }
      endif
   endwhile
   {SOLUSI(S) or C = {} }

   if SOLUSI(S) then    { solusi sudah lengkap }
     return S
   else
     write('tidak ada solusi')
   endif
```

Gambar 2. Skema umum algoritma greedy

Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag1.pdf

Jika ditinjau pada beberapa kasus, dapat diperhatikan bahwa beberapa kasus tidak menghasilkan solusi optimum jika diselesaikan dengan algoritma greedy. Optimum global yang dihasilkan pada beberapa permasalahan merupakan solusi sub-optimum atau pseudo-optimum. Alasan yang pertama adalah algoritma greedy tidak mengecek semua kemungkinan yang ada sehingga ada kemungkinan solusi optimum yang terlewat. Kedua, fungsi seleksi yang dimiliki satu permasalahan sering kali memiliki banyak variansi sehingga harus dipilih fungsi seleksi yang tepat jika ingin semakin mendekati solusi optimal. Algoritma greedy akan sangat cocok pada permasalahan yang tidak memerlukan solusi optimum yang mutlak dan lebih mementingkan kecepatan waktu.

B. Sistem Operasi

Sistem operasi merupakan salah satu komponen struktur sistem komputer. Komponen struktur sistem komputer adalah sebagai berikut:

1. Perangkat keras

   Menyediakan sumber daya dasar untuk komputasi (I/O, CPU, Memory)

2. Sistem operasi

   Mengkoordinasi penggunaan perangkat keras antara beberapa pengguna

3. Program aplikasi

   Mendefinisikan cara penggunaan perangkat keras untuk membantu menyelesaikan masalah pengguna

4. Pengguna

   Subjek yang akan menggunakan komputer (orang, mesin, komputer lainnya)

Sistem operasi adalah program yang bertugas sebagai perantara antara pengguna komputer dengan perangkat keras komputer. Sistem operasi memiliki tujuan sebagai berikut:

1. Mengeksekusi program pegguna dan mempermudah pemecahan masalah pengguna
2. Membuat komputer dapat digunakan

3. Menggunakan perangkat keras komputer dengan efisien

Berdasarkan tujuan di atas, dapat disimpulkan bahwa sistem operasi yang baik harus dapat menjamin semua aplikasi yang ada di dalam komputer dapat menjalankan tugas yang mereka miliki dengan lancar. Jika ada beberapa aplikasi yang berjalan bersamaan, sebuah sistem operasi diharapkan dapat mengatur jadwal pemakaian hardware dengan tepat agar proses yang dijalankan sebisa mungkin tidak mengganggu satu sama lain. Secara umum, sistem operasi terdiri atas beberapa bagian:

1. Boot

   Berfungsi untuk meletakkan kernel ke dalam memory

2. Kernel

   Inti dari sebuah sistem operasi, dapat mengontrol segala hal di dalam sistem

3. Shell

   Membaca input dari user

4. Library

   Menyediakan fungsi standar yang dapat digunakan oleh aplikasi lain

5. Driver

   Media interaksi dengan perangkat keras

Beberapa layanan inti yang dapat diberikan oleh sistem operasi kepada perangkat lunak di antaranya adalah akses ke disk, manajemen memori, *disk scheduling*, dan UI. Pada bagian berikutnya akan dijelaskan mengenai manajemen memori.

C. Manajemen Memori

Secara singkat, manajemen memori adalah proses untuk mengontrol memori komputer dengan menempatkan memori yang diperlukan oleh sebuah program dan membebaskannya ketika sudah tidak digunakan lagi. Pada dasarnya, untuk sebuah program atau proses dapat dijalankan, program tersebut harus dimasukkan ke dalam memori. Akan ada antrian program yang menunggu untuk dimasukkan ke dalam memori untuk dieksekusi. Berikut adalah tahapan-tahapan agar program dapat dieksekusi:

1. Address Binding

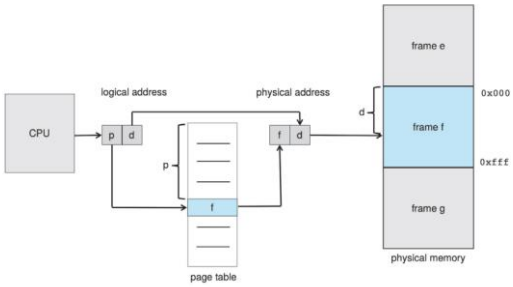
   Address binding adalah proses untuk memetakan suatu alamat di dalam memori ke alamat lainnya. Alamat dibagi menjadi dua, yaitu alamat fisik dan alamat virtual. Alamat fisik adalah alamat yang merujuk pada lokasi sebenarnya di dalam unit memori sedangkan alamat virtual adalah alamat yang dihasilkan oleh CPU ketika eksekusi dijalankan

2. Swapping

   Swapping adalah suatu proses untuk mengeluarkan memori secara sementara ke dalam sebuah *backing store* yang kemudian akan dimasukkan lagi jika ingin digunakan kembali. Swapping menggunakan algoritma *roll out, roll in* sehingga proses yang punya prioritas rendah akan di-*swap out* dan digantikan oleh proses yang memiliki prioritas tinggi

3. Paging

   Proses untuk memecah memori fisik menjadi frame dan memecah memori virtualk menjadi page sehingga ketika proses eksekusi dijalankan, page akan diload ke frame yang tersedia



Gambar 3. Proses paging

Sumber: os-book chapter 9

4. Alokasi memori kontigu

D. Alokasi Memori Kontigu

Memori utama pada dasarnya harus dapat memenuhi kebutuhan dari sistem operasi serta proses-proses yang dipanggil oleh pengguna. Karena keterbatasan memori, maka memori haruslah dialokasikan secara efisien. Memori utama ini biasanya dibagi menjadi dua bagian, kebutuhan dari sistem operasi biasanya diletakkan pada memori rendah sedangkan proses yang dapat dipanggil oleh user diletakkan pada memori tinggi.

Terdapat dua jenis alokasi memori, *single-partition allocation* dan *multiple-partition allocation*. *Single-partition allocation* biasa digunakan untuk melakukan proteksi terhadap memori sedangkan *multiple-partition allocation* biasa dilakukan untuk mengalokasikan memori. Untuk melakukan alokasi memori, caranya adalah memisahkan proses ke dalam partisi-partisi dengan ukuran yang bervariasi dengan satu partisi memiliki satu proses.

Sebuah *hole* adalah sebagian memori yang tersebar di seluruh memori. Sistem operasi akan melacak hole mana saja yang tersedia dan sudah terpakai. Ketika ada proses, sistem operasi akan menempatkannya kepada *hole* yang cukup untuk menampung proses tersebut. *Hole* ini dapat dialokasikan sehingga jika ada proses yang tidak memakai semua kapasitas *hole* maka sisa dari ukuran *hole* dapat dipakai untuk membuat *hole* baru. *Hole* yang dipartisi akan digabungkan lagi jika proses sudah selesai.

Ada 3 cara untuk melakukan alokasi *hole*:

1. First fit

   Mengalokasikan *hole* yang pertama apabila cukup

2. Best fit

Mengalokasikan *hole* yang menyisakan ukuran paling sedikit

3. Worst fit

   Mengalokasikan *hole* yang menyisakan ukuran paling besar

Berdasarkan karakteristiknya, first fit merupakan algoritma dengan utilisasi waktu paling sedangkan best fit merupakan algoritma dengan utilisasi ruang penyimpanan paling baik.

Ketika melakukan alokasi akan ada terjadi kemungkinan fragmentasi, baik internal maupun external. Fragmentasi internal terjadi apabila memori yang dialokasikan lebih besar daripada memori yang diminta. Fragmentasi external terjadi apabila total memori cukup untuk mengakomodasi tapi tempat yang dimiliki tidak kontigu. Worst fit ada untuk meminimalisir efek dari fragmentasi external.

III. Analisis dan Pembahasan

Pada makalah ini, permasalahan yang harus dipecahkan adalah jika diberikan sebuah memori yang memiliki ukuran *hole* yang sudah terurut berdasarkan letaknya maka tentukan *hole* mana saja yang akan digunakan untuk mengakomodasi permintaan memori yang memiliki beberapa ukuran tertentu. Permasalahan ini akan diselesaikan menggunakan 3 metode, yaitu first fit, best fit, serta worst fit.

Secara umum, formulasi yang untuk menyelesaikan permaslahan ini adalah sebagai berikut:

a. Total *hole* yang terdapat pada memori: n

b. Total permintaan memori: m

c. Himpunan *hole* yang dapat dialokasikan (blok memori yang tersedia): [h1, h2, h3, h4 … hn] merupakan array yang berisi ukuran *hole*

d. Himpunan permintaan memori: [r1, r2, r3, r4, …, rm] merupakan array yang berisi ukuran permintaan memori

e. Himpunan solusi: [s1, s2, s3, s4, …, sm] merupakan array dengan elemen si dengan i adalah urutan permintaan memori dan si adalah letak alokasi pada permintaan terkait

A. Program Uji

Secara singkat, manajemen memori adalah proses untuk mengontrol memori komputer dengan menempatkan memori yang diperlukan

1. First fit

Berdasarkan penjelasan pada landasan teori, ide dari mengerjakan alokasi memori dengan menggunakan first fit adalah menelusuri *hole* yang diberikan lalu mengalokasikannya dengan permintaan memori apabila *hole* cukup besar. Menggunakan metode ini memiliki tujuan untuk meminimalisir waktu yang dibutuhkan untuk alokasi memori.

Berikut ini adalah elemen-elemen greedy untuk metode first fit:

1. Himpunan kandidiat

   Berisikan *hole* yang sudah diurutkan letaknya di dalam memori

2. Himpunan solusi (s)

   Berisikan lokasi *hole* yang dipilih

3. Fungsi solusi

   Memeriksa apakah semua permintaan memori sudah terakomodasi atau tidak, jika ada yang tidak terakomodasi maka tidak terdapat solusi menggunakan metode terkait

4. Fungsi seleksi

   Memilih *hole* dengan letak paling awal

5. Fungsi kelayakan

   Memeriksa apakah *hole* yang akan dipilih lebih besar daripada permintaan atau tidak

6. Fungsi obyektif

   Meminimalkan waktu yang diperlukan untuk melakukan alokasi memori

Langkah-langkah untuk menyelesaikan permasalahan alokasi memori dengan metode first fit adalah sebagai berikut:

1. Buatlah sebuah list untuk menyimpan letak alokasi memori dengan menginisiasikannya dengan ukuran sebesar ukuran proses yang diminta dan elemennya dengan -1
2. Lakukan pencarian secara sekuensial dari indeks pertama pada blok memori yang tersedia
3. Jika terdapat *hole* yang cukup besar untuk menampung proses maka letakkan proses pada *hole* tersebut
4. Keluar dari iterasi blok memori dan ulangi untuk setiap proses yang diminta

Berikut ini adalah potongan kode untuk metode first fit dengan menggunakan bahasa pemrograman python:

```python
import timeit
def First_fit(hole_size, process_size):
    m = len(hole_size)
    n = len(process_size)
    allocate = [-1] * n
    start_time = timeit.default_timer()
    for i in range(n):
        for j in range(m):
            if hole_size[j] >= process_size[i]:
                allocate[i] = j
                hole_size[j] -= process_size[i]
                break
    end_time = timeit.default_timer()

    print("Proses ke-\tUkuran Proses\t Hole ke-")
    for i in range(n):
        print(str(i + 1) + "\t\t" + str(process_size[i]) + "\t\t", end=" ")

        if allocate[i] != -1:
            print(allocate[i] + 1)
        else:
            print("Tidak Teralokasikan")
    print("Waktu eksekusi: ", 1000*(end_time - start_time) , " ms")
```

Gambar 4. Potongan kode metode first fit

Sumber: Dokumentasi Pribadi

Kompleksitas algoritma greedy di atas dapat dianalisis berdasarkan jumlah pengecekan *hole*. Dapat dilihat bahwa untuk setiap permintaan memori, akan diiterasi semua *hole* yang tersedia hingga ditemukan *hole* yang ukurannya cukup besar. Maka dari itu, kompleksitas algoritma ini jika dinyatakan sebagai notasi big-O adalah sebagai berikut:

$$O(\text{mn})$$

Serta kasus terbaiknya adalah sebagai berikut:

$$O(\text{m})$$

Kasus ini dapat terjadi apabila *hole* pertama cukup besar untuk menampung semua permintaan memori.

2. Best fit

Berdasarkan penjelasan pada landasan teori, ide dari mengerjakan alokasi memori dengan menggunakan best fit adalah menelusuri *hole* yang diberikan lalu mengalokasikannya dengan permintaan memori dengan mencari *hole* yang cukup serta selisih antara permintaan dengan ukuran *hole* paling kecil. Menggunakan metode ini memiliki tujuan untuk mengoptimalisasi ruang penyimpanan sehingga mencegah terjadinya fragmentasi internal.

Berikut ini adalah elemen-elemen greedy untuk metode best fit:

1. Himpunan kandidiat

   Berisikan *hole* yang sudah diurutkan letaknya di dalam memori

2. Himpunan solusi (s)

   Berisikan lokasi *hole* yang dipilih

3. Fungsi solusi

   Memeriksa apakah semua permintaan memori sudah terakomodasi atau tidak, jika ada yang tidak terakomodasi maka tidak terdapat solusi menggunakan metode terkait

4. Fungsi seleksi

   Memilih *hole* dengan selisih antara permintaan dengan ukuran *hole* paling kecil

5. Fungsi kelayakan

   Memeriksa apakah *hole* yang akan dipilih lebih besar daripada permintaan atau tidak

6. Fungsi obyektif

   Memaksimalkan utilisasi ruang memori

Langkah-langkah untuk menyelesaikan permasalahan alokasi memori dengan metode best fit adalah sebagai berikut:

1. Buatlah sebuah list untuk menyimpan letak alokasi memori dengan menginisiasikannya dengan ukuran sebesar ukuran proses yang diminta dan elemennya dengan -1
2. Lakukan pencarian secara sekuensial dari indeks pertama pada blok memori yang tersedia
3. Jika terdapat *hole* yang cukup besar untuk menampung proses maka lanjutkan proses untuk mencari ukuran *hole* terkecil yang dapat dialokasikan untuk proses terkait
4. Ulangi langkah 2 untuk setiap permintaan memori

Berikut ini adalah potongan kode untuk metode best fit dengan menggunakan bahasa pemrograman python:

```python
def Best_fit(hole_size, process_size):
    m = len(hole_size)
    n = len(process_size)
    allocate = [-1] * n
    start_time = timeit.default_timer()
    for i in range(n):
        for j in range(m):
            if hole_size[j] >= process_size[i]:
                if allocate[i] == -1 or hole_size[allocate[i]] > hole_size[j]:
                    allocate[i] = j
        hole_size[allocate[i]] -= process_size[i]
    end_time = timeit.default_timer()

    print("Proses ke-\tUkuran Proses\t Hole ke-")
    for i in range(n):
        print(str(i + 1) + "\t\t" + str(process_size[i]) + "\t\t", end=" ")

        if allocate[i] != -1:
            print(allocate[i] + 1)
        else:
            print("Tidak Teralokasikan")
    print("Waktu eksekusi: ", 1000*(end_time - start_time) , " ms")
```

Gambar 5. Potongan kode metode best fit

Sumber: Dokumentasi Pribadi

Kompleksitas algoritma greedy di atas dapat dianalisis berdasarkan jumlah pengecekan *hole*. Dapat dilihat bahwa untuk setiap permintaan memori, akan diiterasi semua *hole* yang tersedia dan akan diulang untuk setiap permintaan memori. Maka dari itu, kompleksitas algoritma ini jika dinyatakan sebagai notasi big-O adalah sebagai berikut:

$$O(\text{mn})$$

Notasi big-O akan sama untuk kasus terbaik dan terburuk karena sifat best fit yang harus mengiterasi semua blok memori agar dapat mencari *hole* terkecil yang mungkin

3. Worst fit

Berdasarkan penjelasan pada landasan teori, ide dari mengerjakan alokasi memori dengan menggunakan worst fit adalah menelusuri *hole* yang diberikan lalu mengalokasikannya dengan permintaan memori dengan mencari *hole* yang cukup serta selisih antara permintaan dengan ukuran *hole* paling besar. Menggunakan metode ini memiliki tujuan untuk meminimalkan efek dari fragmentasi eksternal.

Berikut ini adalah elemen-elemen greedy untuk metode worst fit:

1. Himpunan kandidiat

   Berisikan *hole* yang sudah diurutkan letaknya di dalam memori

2. Himpunan solusi (s)

   Berisikan lokasi *hole* yang dipilih

3. Fungsi solusi

Memeriksa apakah semua permintaan memori sudah terakomodasi atau tidak, jika ada yang tidak terakomodasi maka tidak terdapat solusi menggunakan metode terkait

4. Fungsi seleksi

   Memilih *hole* dengan selisih antara permintaan dengan ukuran *hole* paling besar

5. Fungsi kelayakan

   Memeriksa apakah *hole* yang akan dipilih lebih besar daripada permintaan atau tidak

6. Fungsi obyektif

   Memaksimalkan utilisasi ruang memori

Langkah-langkah untuk menyelesaikan permasalahan alokasi memori dengan metode best fit adalah sebagai berikut:

1. Buatlah sebuah list untuk menyimpan letak alokasi memori dengan menginisiasikannya dengan ukuran sebesar ukuran proses yang diminta dan elemennya dengan -1
2. Lakukan pencarian secara sekuensial dari indeks pertama pada blok memori yang tersedia
3. Jika terdapat *hole* yang cukup besar untuk menampung proses maka lanjutkan proses untuk mencari ukuran *hole* terbesar yang dapat dialokasikan untuk proses terkait
4. Ulangi langkah 2 untuk setiap permintaan memori

Berikut ini adalah potongan kode untuk metode worst fit dengan menggunakan bahasa pemrograman python:

```
def Worst_fit(hole_size, process_size):
    m = len(hole_size)
    n = len(process_size)
    allocate = [-1] * n
    start_time = timeit.default_timer()
    for i in range(n):
        for j in range(m):
            if hole_size[j] >= process_size[i]:
                if allocate[i] == -1 or hole_size[allocate[i]] < hole_size[j]:
                    allocate[i] = j
        hole_size[allocate[i]] -= process_size[i]
    end_time = timeit.default_timer()

    print("Proses ke-\tUkuran Proses\t Hole ke-")
    for i in range(n):
        print(str(i + 1) + "\t\t" + str(process_size[i]) + "\t\t", end=" ")

        if allocate[i] != -1:
            print(allocate[i] + 1)
        else:
            print("Tidak Teralokasikan")
    print("Waktu eksekusi: ", 1000*(end_time - start_time) , " ms")
```

Gambar 6. Potongan kode metode worst fit

Sumber: Dokumentasi Pribadi

Kompleksitas algoritma greedy di atas dapat dianalisis berdasarkan jumlah pengecekan *hole*. Dapat dilihat bahwa untuk setiap permintaan memori, akan diiterasi semua *hole* yang tersedia lalu akan diulang untuk setiap permintaan memori. Maka dari itu, kompleksitas algoritma ini jika dinyatakan sebagai notasi big-O adalah sebagai berikut:

$$O(mn)$$

Notasi big-O akan sama untuk kasus terbaik dan terburuk karena sifat best fit yang harus mengiterasi semua blok memori agar dapat mencari *hole* terkecil yang mungkin.

B. *Pengujian*

1. Lingkungan Pengujian

Processor : AMD Ryzen 7 4700U 2GHz

Ram : 16 GB

OS : Windows 11

2. Hasil Pengujian

Pengujian pertama akan menggunakan data uji sebagai berikut:

Blok alokasi memori (*hole* tersedia) = 100, 50, 30, 120, 35

Permintaan memori = 20, 60, 70, 40

Berikut adalah hasil pengujiannya

```
Hasil Pengujian First Fit
Proses ke-      Ukuran Proses    Hole ke-
1               20               1
2               60               1
3               70               4
4               40               2
Waktu eksekusi:  0.005800000000000249  ms
Hasil Pengujian Best Fit
Proses ke-      Ukuran Proses    Hole ke-
1               20               3
2               60               1
3               70               4
4               40               1
Waktu eksekusi:  0.010300000000018628  ms
Hasil Pengujian Worst Fit
Proses ke-      Ukuran Proses    Hole ke-
1               20               4
2               60               1
3               70               4
4               40               2
Waktu eksekusi:  0.00880000000000325  ms
```

Gambar 7. Hasil pengujian TC-1

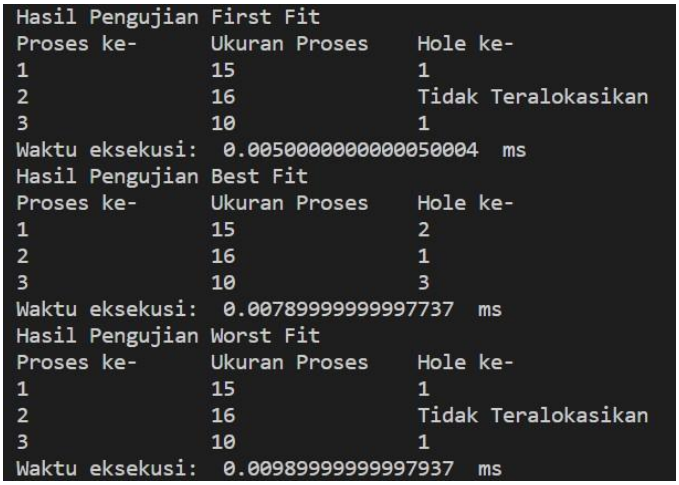
Sumber: Dokumentasi Pribadi

Pengujian kedua akan menggunakan data uji sebagai berikut:

Blok alokasi memori (*hole* tersedia) = 30, 15, 10

Permintaan memori = 15, 16, 10

Berikut adalah hasil pengujiannya



Gambar 8. Hasil pengujian TC-2

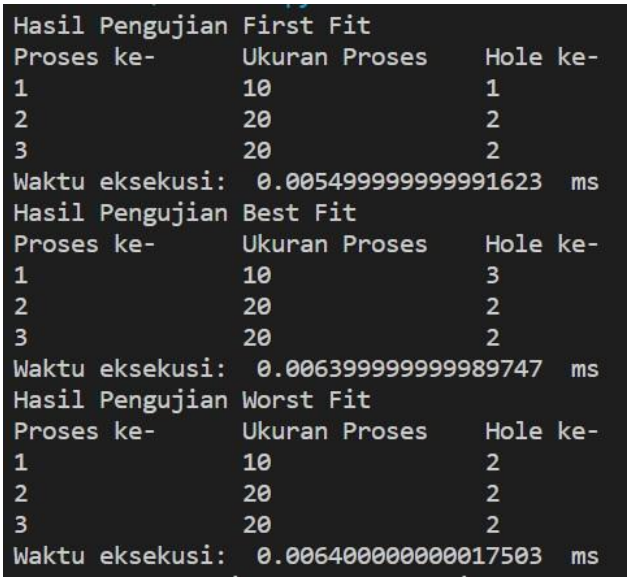
Sumber: Dokumentasi Pribadi

Pengujian kedua akan menggunakan data uji sebagai berikut:

Blok alokasi memori (*hole* tersedia) = 15, 50, 10

Permintaan memori = 10, 20, 20

Berikut adalah hasil pengujiannya



Gambar 9. Hasil pengujian TC-3

Sumber: Dokumentasi Pribadi

C. Pembahasan

Berdasarkan Pengujian yang telah dilakukan dapat dilihat bahwa pada ketiga kasus pengujian, hasil pengujian first fit memiliki waktu eksekusi paling cepat di antara dua metode lainnya. Pada kasus uji pertama, dapat dilihat selisih waktu antara first fit dengan worst fit dan best fit hamper mencapai 2 kali lipat. Metode worst fit dan best fit memiliki waktu eksekusi yang mirip dalam 3 percobaan dan perbedaan tidak terlalu signifikan karena mengaharuskan untuk mengiterasi semua blok memori. Hal ini sesuai dengan analisis pada bagian A. Meskipun dengan kompleksitas waktu yang sama, metode first fit cenderung memiliki waktu eksekusi yang lebih cepat karena adanya penghentian iterasi di tengah jalan.

Dapat dilihat pada kasus uji kedua, penggunaan metode first fit dan worst fit menghasilkan adanya proses yang tidak dialokasikan ke dalam *hole*. Pada penggunaan metode best fit, semua proses dapat dialokasikan ke dalam *hole*. Hal ini sesuai dengan tujuan metode best fit, yaitu untuk memaksimalkan utilisasi ruang penyimpanan dengan meminimalisir fragmentasi internal. Hal ini dapat terjadi karena metode best fit akan memprioritaskan alokasi memori dengan ukuran *hole* paling pas dengan permintaan memori. Dengan begitu utilisasi ruang penyimpanan dapat lebih optimal. Dari pengujian ini juga dapat dibuktikan meskipun optimal dari segi waktu, penggunaan metode first fit memiliki kemungkinan adanya proses yang tidak dapat dialokasikan.

Pada pengujian yang ketiga, dapat dilihat bahwa dengan menggunakan metode first fit, proses 1 dialokasikan ke *hole* 1 sedangkan proses 2 dan 3 dialokasikan ke *hole* 2. Pada penggunaan metode best fit didapat alokasi proses 1 ke *hole* 1 sedangankan proses 2 dan 3 dialokasikan ke *hole* 2. Berbeda dengan metode worst fit, proses 1, 2 dan 3 dialokasikan ke dalam *hole* yang sama, yaitu *hole* 2. Dapat dilihat bahwa pada penggunaan metode first fit dan best fit terjadi external fragmentation, yaitu alokasi memori tidak kontigu sedangkan pada penggunaan metode worst fit tidak terjadi external fragmentation. Hal ini dapat terjadi karena metode worst fit akan memprioritaskan *hole* yang memiliki ukuran paling besar agar dapat memuat proses yang memiliki ukuran lebih kecil.

Berdasarkan pengujian, masing-masing metode memiliki kelebihan dan kekurangannya masing-masing. Hasil pengujian dilakukan dengan mengambil data dengan ukuran yang relatif kecil maka perbedaan hasil dari masing-masing metode tidak begitu signifikan terlihatnya. Pada implementasi aslinya, dengan ukuran alokasi memori serta permintaan memori yang lebih besar pasti akan ada beberapa kasus yang lebih unik dan memperlihatkan perbedaan hasil yang lebih jauh lagi.

IV. KESIMPULAN

Algoritma greedy adalah algoritma yang mengambil optimum lokal pada setiap langkahnya dengan harapan mendapatkan optimum global. Pada permasalahan alokasi memori kontigu, hasil optimal memiliki arti yang berbeda-beda sehingga dengan mengganti fungsi seleksi dapat dihasilkan optimal yang berbeda. Berdasarkan pengujian yang telah dilakukan, dengan melihat pada kasus uji 1, dapat disimpulkan bahwa metode first fit cenderung menghasilkan waktu eksekusi yang paling optimal dibandingkan dengan metode best fit serta worst fit. Pada kasus uji 2, dapat dilihat bahwa utilisasi ruang penyimpanan yang dihasilkan oleh metode best fit merupakan yang paling optimal ketimbang metode first fit dan worst fit. Pada kasus uji 3, dapat dilihat bahwa metode worst fit dapat meminimalisir kasus terjadinya fragmentasi eksternal.

Untuk penelitian lanjutan makalah ini, diharapkan agar mencoba memperbesar kasus uji dan mencoba menggunakan atau menggabungkan strategi algoritma lainnya agar dihasilkan hasil yang lebih optimal lagi.

LINK VIDEO

https://youtu.be/KTWvtd6v98g

UCAPAN TERIMA KASIH

Penulis mengucapkan puji dan syukur kepada Tuhan Yang Maha Esa karena atas berkat dan rahmat-Nya penulis diberikan kesehatan dan pikiran agar mampu menyelesaikan makalah ini tepat waktu tanpa hambatan yang begitu berarti. Penulis mengucapkan terima kasih sebesar-besarnya kepada Pak Rinaldi Munir sebagai dosen pengampu mata kuliah IF2211, Strategi Algoritma beserta staf pengajar serta asisten yang telah mengajarkan penulis selama satu semester. Penulis juga ingin mengucapkan terima kasih kepada keluarga dan teman yang telah memberikan dukungan untuk menyelesaikan makalah ini.

REFERENCES

[1] Munir, Rinaldi. 2021. Algoritma Greedy (Bagian 1). https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag1.pdf. Diakses pada 19 Mei 2022.

[2] GeeksforGeeks. 2021. Best-Fit Allocation in Operating System. https://www.geeksforgeeks.org/best-fit-allocation-in-operating-system/?ref=lbp. Diakses pada 20 Mei 2022.

[3] GeeksforGeeks. 2021. Worst-Fit Allocation in Operating System. https://www.geeksforgeeks.org/worst-fit-allocation-in-operating-systems/?ref=lbp. Diakses pada 20 Mei 2022.

[4] Prepinsta. 2022. First Fit Best Fit Worst Fit in OS. https://prepinsta.com/operating-systems/first-fit-best-fit-worst-fit-in-os-example/. Diakses pada 20 Mei 2022.

[5] Silberschatz, Avi. Baer Galvin, Peter. Gagne, Greg, Operating System Concepts, 10th ed. John Wiley & Sons: Hoboken, 2018.

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, 22 Mei 2022

Kevin Roni 13520114