Penerapan Algoritma *Greedy* Pada Penjadwalan *Autonomous Vehicle* Saat Berada di Persimpangan Jalan

Azhar Abdurrasyid - 13517097
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13517097@std.stei.itb.ac.id

***Abstract*—Perkembangan *Autonomous Vehicle* meningkat sangat pesat dalam dekade ini. Dengan begitu, cepat atau lambat seluruh kendaraan di dunia akan dipenuhi dengan *Autonomous Vehicle.* Dalam keadaan tersebut, suatu kendaraan akan mengetahui posisi kendaraan lain bukan dari sensor melainkan dari *server*. Idealnya, kendaraan tidak akan lagi berhenti untuk menunggu kendaraan lain, hanya mengurangi kecepatan. Bahkan saat di persimpangan, yang pada umumnya merupakan posisi dimana kendaraan harus berhenti untuk menunggu kendaraan lainnya, *Autonomous Vehicle* tidak akan berhenti melainkan hanya mengatur kecepatannya agar tidak terjadi kecelakaan. Makalah ini bertujuan untuk menerapkan algoritma *greedy* untuk penjadwalan dan pengaturan kecepatan *Autonomous Vehicle* di persimpangan jalan.**

***Kata Kunci—Greedy; Autonomous Vehicle;* Penjadwalan*;* Pengaturan Kecepatan**

I. PENDAHULUAN

Autonomous Vehicle adalah sebuah kendaraan yang memiliki fitur *Self-Driving* sehingga pengguna hanya perlu untuk menentukan destinasi perjalanan dan sisanya akan dilakukan oleh fitur tersebut. Eksperimen mengenai fitur ini sudah mulai dilakukan sejak 1 abad yang lalu, tepatnya pada tahun 1920. Baru pada tahun 1984, *Navlab* milik *Carneige Mellon University* berhasil membuat mobil yang sepenuhnya dikendalikan otomatis. Dilanjutkan pada tahun yang sama oleh Meceds-Benz. Sejak saat itu banyak perusahaan besar yang mulai mengembangkan fitur tersebut seperti General Motors, Continental Automotive Sytems, Nissan, Toyota, dan bahkan Google.

Penemuan terbaru terkait dengan *Autonomous Vehicle* dilakukan oleh salah satu perusahaan mobil listrik terbesar di dunia yaitu Tesla. Pada tanggal 23 April 2019, Tesla baru saja mengeluarkan *chip* terbarunya yang diklaim sebagai kunci unutk fitur ‘Full Self-Driving’ di masa depan. Kendaraan yang menggunakkan *chip* tersebut akan dilengkapi dengan 8 kamera, 12 sensor ultra sonik, dan radar. Dengan klaim seperti itu, *chip* tersebut masih sering mengalami kegagalan saat berada di persimpangan jalan.

Masalah utama yang dihadapi peneliti *Autonomous Vehicle* adalah untuk mengetahui keadaan sekitar dan posisi kendaraan lain di jalan. Saat ini, sebagian besar peneliti selalu mengandalkan berbagai macam sensor salah satunya adalah *Lidars.* Sensor ini diklaim akan memberikan data tiga dimensi mengenai lingkungan sekitar. Walaupun terdengar sangat canggih tapi masih banyak yang meragukan sensor tersebut. Namun, jika semua kendaraan merupakan *Autonomous Vehicle*, dibanding harus mengambil data lingkungan sekitar dengan sensor yang ada, akan lebih efisien jika semua kendaraan dapat berkomunikasi satu sama lain melalui *server*. Dengan begitu masing-masing kendaraan akan mengetahui posisi, jalur apa yang akan dilalui, dan kapan jalur tersebut akan dilalui kendaraan lain.

Masalah yang muncul kemudian adalah bagaimana cara *server* menangani *request* dari masing-masing kendaraan. Masalah lain yang akan muncul terutama di persimpangan jalan adalah ketika jalur kendaraan satu berpotongan dengan jalur kendaraan lainnya. Salah satu solusi terhadap masalah tersebut adalah dengan menggunakkan algoritma *greedy.* Algoritma *greedy* akan mengatur penjadwalan *request* dengan menyesuaikan waktu *request* kendaraan. Penjadwalan ini diharapkan akan meminimalisir *delay* waktu kendaraan lain.

II. TEORI DASAR

A. Algoritma Greedy

[3]Algoritma *greedy* mungkin merupakan metode yang paling populer untuk memecahkan masalah optimisasi. Algoritma ini sederhana dan langsung ke tujuan (*straight to the point)*. Secara bahasa *greedy* artinya ‘rakus’ atau ‘tamak’ yang cenderung berkonotasi negatif. Prinsip utama yang digunakan algoritma *greedy* adalah “*take what you can get now!”*.

Algoritma *greedy* membentuk solusi langkah per langkah. Terdapat banyak pilihan yang perlu dieksplorasi pada setiap langkah solusi. Oleh karena itu, pada setiap langkah harus dibuat keputusan yang terbaik dalam menentukan pilihan. Keputusan yang telah diambil pada suatu langkah tidak dapat

diubah lagi pada langkah selanjutnya. Hal ini menyebabkan algoritma ini cederung memiliki efisiensi waktu dibanding algoritma optimisasi lain. Pendekatan yang digunakan dalam algoritma *greedy* adalah membuat pilihan yang terlihat memberikan perolehan terbaik, dengan kata lain algortima ini akan memilih solusi optimum lokal dengan harapan akan mendekati optimum global.

Kebanyakan persoalan optimasi yang menggunakkan algoritma *greedy* memiliki beberapa elemen sebagai berikut:

1. *Himpunan kandidat (C)*
   Himpunan ini berisi elemen-elemen penyusun solusi.

2. *Himpunan solusi (S)*

   Himpunan ini berisi kandidHimpunan ini berisi kandidat-kandidat yang terpilih sebagai solusi persoalan. Dengan kata lain, himpunan solusi adalah himpunan bagian dari himpunan kandidat.

3. Fungsi seleksi

   Fungsi ini digunakan pada setiap langkah untuk memilih kandidat yang paling memungkinkan mencapai solusi optimal, dalam hal ini yang dimaksud adalah optimum lokal. Kandidat yang sudah dipilih akan langsung dihapus dari himpunan kandidat sehingga kandidat tersebut tidak akan terpilih lebih dari satu kali.

4. Fungsi kelayakan (*feasible)*

   Fungsi ini akan memeriksa apakah suatu kandidat yang telah dipilih dapat memberikan solusi yang layak, dalam hal ini kandidat tersebut bersama-sama dengan himpunan solusi yang sudah terbentuk tidak melanggar batasan-batasan yang ada. Kandidat yang dianggap layak akan dimasukkan ke dalam himpunan solusi sedangkan yang tidak layak akan dihapus dari himpunan kandidat.

5. *Fungsi Obyektif*
   Fungsi ini akan menetukan parameter optimal dari nilai solusi

Secara umum, algoritma *greedy* dapat digambarkan sebagai berikut:

```
1. inisialisasi himpunan solusi dengan kosong
2.pilih sebuah kandidat dengan menggunakkan fungsi seleksi dari himpunan kandidat
3. hapus elemen yang terpilih di langkah ke-2 dari himpunan kandidat
4. periksa apakah kandidat yang dipilih jika digabungkan dengan himpunan solusi akan membentuk solusi yang layak atau tidak.
5. jika langkah-4 mengembalikan nilai 'ya' maka masukkan kandidat tersebut ke dalam himpunan solusi. jika tidak, kandidat tersebut tidak akan dipertimbangkan lagi
6. periksa apakah himpunan solusi telah memberikan solusi yang lengkap menggunakkan fungsi obyektif.
7. jika langkah-6 mengembalikan nilai 'ya' maka berhenti. jika tidak, lakukan langah-2
```

Pada beberapa masalah optimasi, algoritma *greedy* tidak selalu memberikan solusi optimum global. Jika jawaban yang optimum global tidak benar-benar diperlukan, maka algoritma *greedy* akan sangat berguna untuk menghasilkan solusi yang mendekati optimum, dibanding menggunakan algoritma kompleks dan kurang efisien dalam waktu untuk menemukan solusi optimum global.

B. Job Scheduling

[2]Secara harfiah *Job Scheduling* berarti penjadwalan pekerjaan atau tugas untuk sebuah sistem yang terdiri dari sejumlah server yang harus memproses atau melayani sejumlah *client.* Hubungan antara *server* dan *client* dapat berupa TCP/IP, *processor* di sistem operasi, ataupun kasir sebagai *server* dan nasabah bank sebagai *client.* Dalam makalah ini yang berlaku sebagai *server* adalah komputer yang terus mengeksekusi program dan menyimpan basis data dan yang berlaku sebagai *client* adalah semua *Autonomous Vehicle.*

Algoritma penjadwalan tugas bertujuan untuk memaksimalkan kinerja *server* dalam hal melayani permintaan *client* sekaligus meminimalisir waktu tunggu *client* dan waktu sistem bekerja. Berdasarkan eksekusinya, algoritma penjadwalan tugas terdiri dari dua jenis. Jenis pertama adalah *non pre-emptive. A*lgoritma jenis ini tidak memperbolehkan sebuah proses yang sedang dijalankan untuk diinterupsi atau diganggu oleh *client* dalam kondisi apapun sampai proses tersebut selesai dieksekusi. Jenis kedua adalah *pre-emptive.* Algoritma ini memperbolehkan sebuah proses yang sedang berjalan diganggu atau diinterupsi oleh *client* walaupun proses tersebut belum selesai. Berdasarkan tenggat waktu, algoritma penjadwalan tugas terbagi menjadi dua yaitu *job without deadline* dan *job with deadline. Job without deadline* adalah proses yang tidak memiliki tenggat waktu, jadi sebuah proses tidak ditentukan harus selesai setelah berapa lama. *Job with deadline* adalah proses yang memiliki tenggat waktu, jika proses tersebut belum selesai sampai tenggat waktu yang ditentukan maka proses itu tidak akan dijalankan lagi.

Ada beberapa kriteria yang menjadi acuan untuk algoritma *job scheduling.* Kriteria tersebut adalah sebagai berikut.

1) Utilization

Utilization adalah persentase penggunaan sumber daya sistem. Optimalnya kriteria ini harus mencapai angka maksimal.

2) Throughput

Throughput adalah banyaknya proses yang diselesaikan persatuan waktu. Optimalnya kriteria ini harus mencapai angka maksimal.

3) Turn around time

Turn around time adalah waktu yang diperlukan unutk mengeksekusi sebuah proses. Optimalnya kriteria ini harus mencapai angka minimal.

4) Waiting time

Waiting time adalah waktu yang diperlukan sebuah proses dari *request* sampai mulai dieksekusi oleh *server.*Optimalnya kriteria ini harus mencapai angka minimal.

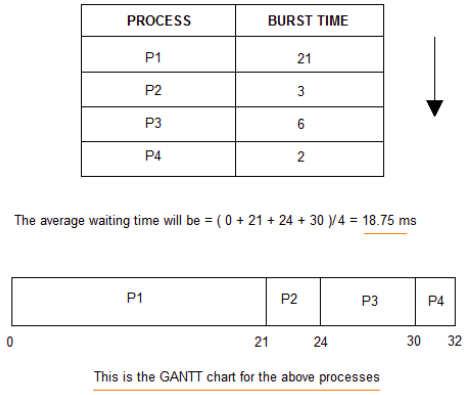
5) Response time

Response time adalah waktu yang dibutuhkan ketika sebuah *request* diterima sampai *server* merespon untuk pertama kalinya. Optimalnya kriteria ini harus mencapai angka minimal.

Terdapat beberapa variasi algoritma dalam persoalan *job scheduling.* Variasi tersebut adalah sebagai berikut.

1) First Come First Serve (FCFS)

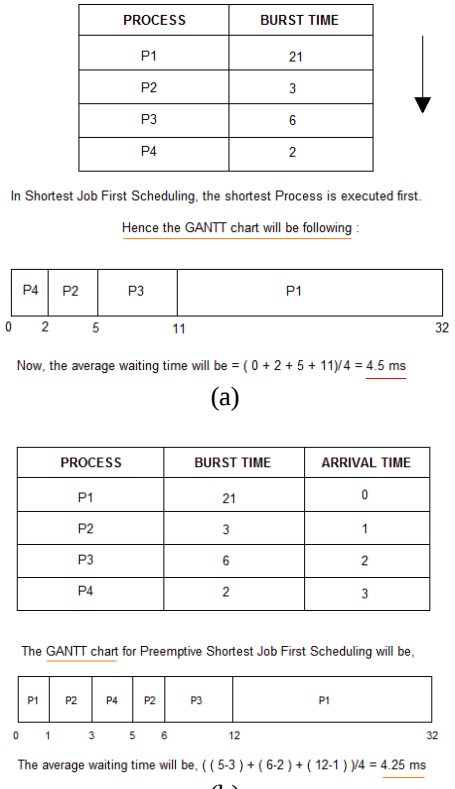
Algoritma ini memanfaatkan strategi *greedy.* Algoritma ini menggunakkan prinsip yang sama dengan struktur data *queue* yaitu *first in first out (FIFO)*. *Server* akan melayani *client* yang datang lebih awal terlebih dahulu. FCFS termasuk ke dalam kategori *non pre-emptive*. Kelebihan menggunakkan algoritma ini adalah karena menggunakkan prinsip yang sama dengan *queue* sehingga akan lebih mudah untuk diimplementasikan. Kekurangan dari algoritma ini yaitu berpotensinya muncul *Convoy Effect* ketika waktu pemrosesan tiap-tiap *request* berbeda. *Convoy Efect* adalah efek yang membuat proses yang memiliki waktu yang lebih singkat akan terhalang oleh proses lain sehingga rata-rata waktu tunggu proses akan lebih besar.



Gambar 1. Viisualisasi Algoritma *FCFS.* Sumber : www.studytonight.com/operating-system/first-come-first-serve

2) Shortest Job First (SJF)

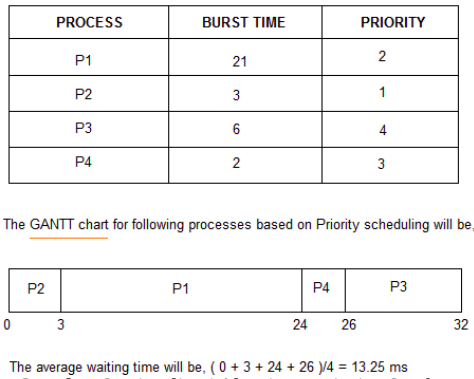
Algoritma ini memanfaatkan strategi *greedy.* Algoritma ini akan menjalankan *request client* berdasarkan perkiraan waktu eksekusi *request* tersebut. Makin cepat waktu eksekusinya maka akan lebih awal dieksekusi. Algoritma ini dapat dilakukan secara *pre-emptive* maupun *non pre-emptive*. Kelebihan dari algoritma ini adalah akan meminimalkan rata-rata *waiting time* ketika waktu kedatangan *request* tersebut bersamaan. Kerugian dari algoritma ini adalah memungkinkan terjadinya *starvation. Starvation* adalah kondisi ketika ada *request* yang tidak pernah dilayani ataupun menunggu terlalu lama untuk dilayani.



Gambar 2. Visualisasi Algoritma SJF (a) *non pre-emptive* (b) *pre-emptive.* Sumber: https://www.studytonight.com/operating-system/shortest-job-first

3) Priority

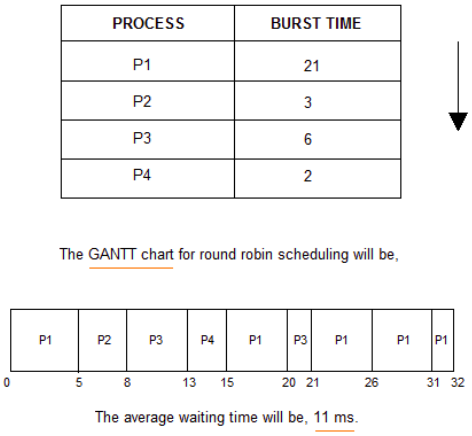
Algoritma ini memanfaatkan strategi *greedy.* Algoritma ini akan melayani *request client* berdasarkan atribut prioritas yang telah ditentukan. *Request* yang memiliki nilai prioritas paling kecil akan dilayani terlebih dahulu. Algoritma ini termasuk kategori *pre-emptive* sehingga proses dapat diinterupsi kapanpun. Kelebihan dari algoritma ini adalah *request* yang dilayani adalah *request* yang memiliki tingkat urgensi yang tinggi.



Gam bar 3. Visualisasi Algoritma *Priority*. Sumber: https://www.studytonight.com/operating-system/priority-scheduling

4) Round Robin Scheduling

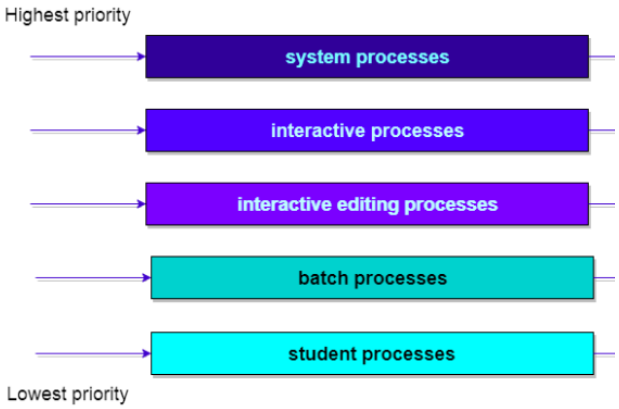
Algoritma ini memanfaatkan strategi *greedy.* Algoritma ini mirip dengan *FCFS* namun memiliki batsan waktu eksekusi yang telah ditetapkan yang disebut *quantum time.* Ketika suatu proses telah dieksekusi selama *quantum time* dan belum selesai, maka proses tersebut kembali ke urutan terakhir. Algoritma ini termasuk kategori *pre-emptive.*



Gambar 4. Visualisasi Algoritma *Round Robin.* Sumber: https://www.studytonight.com/operating-system/round-robin-scheduling

5) Multilevel Queue Scheduling

Seperti namanya, algoritman ini berprinsip sama seperti *queue* yaitu *first in first out.* Berbeda dengan *FCFS,* algoritma ini tidak membuat antrian untuk tiap prosesnya melainkan membuat antrian untuk tiap kateogri proses. Algoritma ini akan mengkategorikan semua *request* dan akan menentukan algoritma yang tepat untuk setiap kategori berdasarkan tingkat prioritas. Algoritma ini merupakan algoritma yang tidak fleksibel karena dapat suatu proses tidak dapat bergerak antara antrian kategori. Kelebihan algortima ini adalah mengurangi waktu *overhead.*



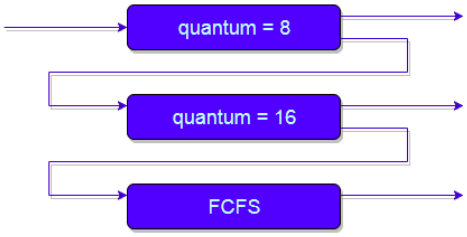
Gambar 5. Visualisasi Algoritma *Multilevel Queue.* Sumber: https://www.studytonight.com/operating-system/multilevel-queue-scheduling

6) Multi-Processor Scheduling

Algoritma ini dikhususkan untuk dijalankan dalam sistem yang memiliki jumlah *server* lebih dari satu. Algoritma ini biasa digunakan pada komputer dengan *multi-core processor*. Algoritma ini memungkinkan untuk melayani *request* secara paralel sebanyak *server* yang tersedia sehingga akan meningkatkan kriteria *throuhput* berkali lipat. Berdasarkan cara kerjanya, algoritma ini terbagi menjadi dua jenis yaitu *symmetric multiprocessing* dan *asymmetric multiprocessing. Symmetric multiprocessing* memberikan kebebasan secara penuh kepada *server* untuk mengatur penjadwalan *request.* Di sisi lain, *asymmetric multiprocessing* menjadikan salah satu *server* sebagai penentu prosedur pemrosesan *request (master)* dan *server* lain harus mengikuti prosedur tersebut (*slave)*.

7) Multilevel Feedback Queue Scheduling

Algoritma ini akan secara permanen memasukkan *request* kedalam salah satu antrian kategori. Algoritma ini mengkategorisasikan sebuah *request* berdasarkan waktu eksekusi *request* tersebut. Jika suatu *request* butuh waktu yang lama untuk dieksekusi maka akan mendapat prioritas lebih rendah dan sebaliknya. Dengan begitu akan mencegah kondisi *starvation*. Kelebihan algoritma ini adalah sangat fleksibel karena sebuah *request* dapat berpindah antrian sesuai atribut prioritas yang dinamis. Walaupun algoritma ini merupakan algoritma yang paling sering dipakai, algoritma ini adalah algoritma yang paling rumit.



Gambar 6. Visualisasi Algoritma *Multilevel Feedback Queue.* Sumber: https://www.studytonight.com/operating-system/multilevel-feedback-queue-scheduling

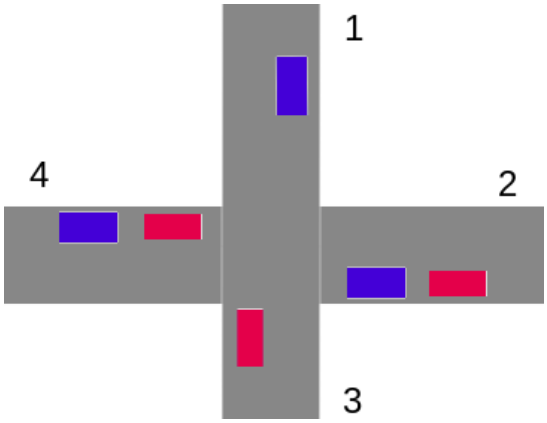
III. Analisa Masalah dan Implementasi Algoritma

A. Permasalahan Penjadwalan Server dan Client

Ada jutaan kendaraan dan ribuan persimpangan jalan di dunia sehingga masalah penjadwalan ini akan terjadi hampir setiap waktu. Dalam konteks ini yang berlaku sebagai *server* adalah komputer yang selalu menjalankan program dan menyimpan seluruh basis data dan *client* adalah kendaraan yang ingin berkomunikasi dengan kendaraan lain. Dengan banyaknya kendaraan di jalan maka sebuah *server* tidak akan cukup untuk menangani seluruh *request.* Oleh karena itu, masalah ini akan diselesaikan dengan pendekatan *multi-processor scheduling.* Dalam permasalahan ini, yang dimaksudnya dengan *request* adalah ketika sebuah kendaraan ingin melewati persimpangan sehingga kendaraan tersebut akan mengirimkan data berupa koordinat-koordinat yang akan dilalui dan sekaligus komponen waktu untuk masing-masing koordinat. Dengan begitu, waktu kedatangan adalah waktu

ketika kendaraan melakukan *request* ke *server* dan waktu proses berakhir adalah ketika kendaraan menerima respon dari *server* berupa himpunan waktu kapan sebuah kendaraan harus berada di koordinat tertenu. Adapun *burst time* adalah selisih dari kedua waktu tersebut. Tidak ada tenggat waktu dalam kasus ini oleh karena itu kasus ini temasuk ke dalam *job without deadline*. *Goal* dari penyelesaian masalah ini adalah untuk meminimalisasi rata-rata *waiting time* dari sejumlah *request* yang ada.

Untuk membandingkan *job scheduling* mana yang akan digunakan untuk masalah ini, penulis akan memodelkan permasalahan ini dalam sebuah lingkungan virtual. Lingkungan ini terdiri dari 6 buah kendaran, 2 buah *server,* dan sebuah persimpangan jalan yang bercabang empat.



Gambar 7. Visualisasi lingkungan virtual.

Berikut merupakan variabel-variabel yang dipakai.

- *Pi: Request* ke-*i*
- *ai:* Waktu kedatangan kendaraan ke-*i* (detik)
- *fi:* Nomor jalan kendaraan ke-*i*
- *ti:* Nomor jalan tujuan kendaraan ke-*i*
- *si:* Nomor server yang dipakai kendaraan ke-*i*
- *bi: Burst time* yang dibutuhkan kendaraan ke-*i* (detik)
- *P timeline:* Nomor *request* pada *timeline*

Pi	*ai*	*fi*	*ti*	*si*	*bi*
1	1	1	2	1	4
2	3	2	4	1	5
3	2	2	1	2	4
4	1	3	2	2	6
5	3	4	3	2	4
6	2	4	2	1	3

Tabel 1. Data kendaraan.

B. Penyelesaian Masalah Menggunakkan Algoritma Greedy

Permasalahan ini termasuk ke dalam kategori *multi-processing scheduling* , *non-preemptive scheduling*, dan *job without deadline.*

1) Langkah Pengerjaan

Untuk menyelesaikan masalah ini, maka akan digunakan algoritma *greedy* dalam masalah *job scheduling* yang sudah ada. Pada kesempatan ini, penulis akan menggunakkan pendekatan *symmetric multi-processing scheduling* dimaan *master server* hanya akan menggunakan algoritma *FCFS* dan *SJF non pre-emptive* karena hanya kedua algoritma itu yang menggunakkan pendekatan *non pre-emptive*.

Berikut adalah rincian langkah pengerjaan.

i. Mensimulasikan data fiktir dengan algoritma *FCFS*

ii. Hitung rata-rata *waiting time*

iii. Mensimulasikan data fiktif dengan algotima *SJF non pre-emptive*

iv. Hitung rata-rata *waiting time*

v. Bandingkan kedua hasil tersebut

2) Hasil dan Analisis

a) Pengerjaan dengan Algoritma FCFS

Berikut adalah algoritma *FCFS* pada *multi-processing scheduling.*

```
1. Bagi daftar kendaraan berdasarkan nomor server (si) yang digunakan
2. Untuk masing-masing daftar kendaraan lakukan:
    2.1 Urutkan dari kecil ke besar daftar kendaraan berdasarkan urutan
     kedatangan (ai)
    2.2 Eksekusi request (ri) berdasarkan daftar yang sudah terurut
```

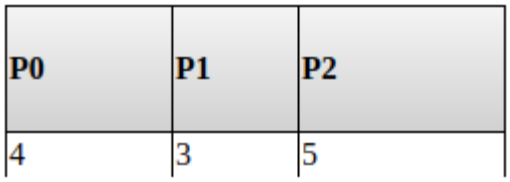
Berikut adalah hasil dari algoritma *FCFS.*

P timeline	*Pi*	*ai*	*fi*	*ti*	*si*	*bi*
0	1	1	1	2	1	4
1	6	2	4	2	1	3
2	2	3	2	4	1	5

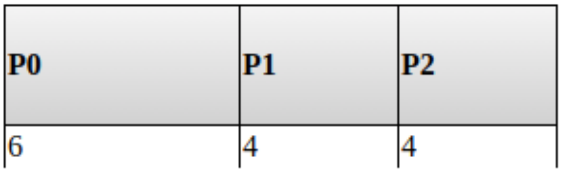
Tabel 2. Data *server* 1 setelah *FCFS*

P timeline	*Pi*	*ai*	*fi*	*ti*	*si*	*bi*
0	4	1	3	2	2	6
1	3	2	2	1	2	4
2	5	3	4	3	2	4

Tabel 3. Data *server* 2 setelah *FCFS*



Gambar 8. *Timeline* Pemrosesan *server* 1



Gambar 9. *Timeline* Pemrosesan *server 2*

Dari data-data tersebut dapat disimpulkan nilai dari beberapa kriteria sebagai berikut.

Avg waiting time	(4 + 7 + 6 + 4)/6 = 3,5
Avg response time	(3 + 5 + 5 + 8)/6 = 3,5
Avg Throughput	1/ [(4+3+5+6+4+4)/6] = 0,2325

Tabel 4. Hasil data

b) Pengerjaan dengan Algoritma SJF

Algoritma *SJF non pre-empitive* pada *multi-processing scheduling*

1. Bagi daftar kendaraan berdasarkan nomor server (si) yang digunakan
2. Untuk masing-masing daftar kendaraan lakukan:
   2.1 Urutkan dari kecil ke besar daftar kendaraan berdasarkan *burst time*
   kedatangan (bi)
   2.2 Eksekusi request (ri) berdasarkan daftar yang sudah terurut

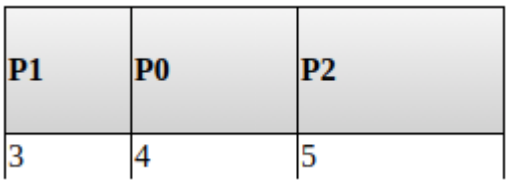
Berikut adalah hasil dari algoritma *SJF.*

P timeline	*Pi*	*ai*	*fi*	*ti*	*si*	*bi*
1	6	2	4	2	1	3
0	1	1	1	2	1	4
2	2	3	2	4	1	5

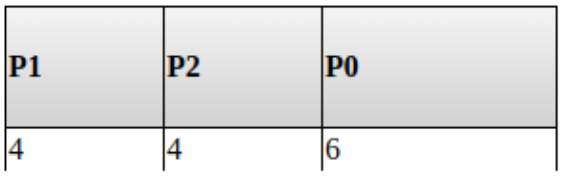
Tabel 5. Data *server* 1 setelah *SJF*

P timeline	*Pi*	*ai*	*fi*	*ti*	*si*	*bi*
1	3	2	2	1	2	4
2	5	3	4	3	2	4
0	4	1	3	2	2	6

Tabel 6. Data *server* 2 setelah *SJF*



Gambar 10. *Timeline* Pemrosesan *server 1*



Gambar 11. *Timeline* Pemrosesan *server 2*

Dari data-data tersebut dapat disimpulkan nilai dari beberapa kriteria sebagai berikut.

Avg waiting time	(3 + 4 + 4 +4)/6 = 3,5
Avg response time	(2 + 5 + 3 + 6)/6 = 2,67
Avg Throughput	1/ [(4+3+5+6+4+4)/6] = 0,2325

Tabel 7. Hasil data

c) Perbandingan FCFS dan SJF

Berikut adalah perbandingan dari algoritma *FCFS* dan *SJF.*

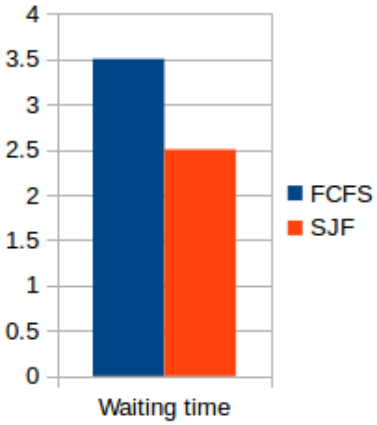


Diagram 1. Perbandingan *Waiting Time*

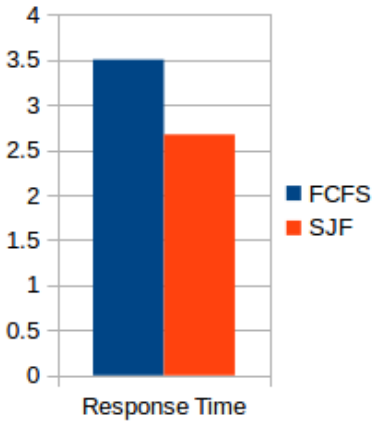


Diagram 2. Perbandigan *Response Time*

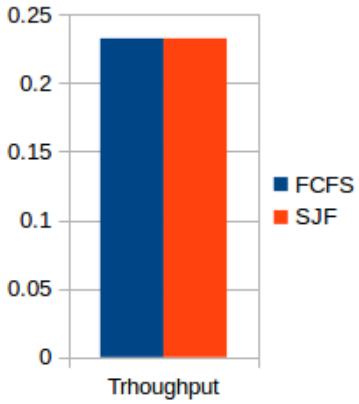


Diagram 3. Perbandingan *Throughput*

IV. Kesimpulan

Manusia mampu bertahan hidup karena manusia terus berinovasi. Inovasi didapatkan ketika seseorang memiliki visi yang jauh dan ingin merealisasikan visi tersebut. Makalah ini adalah salah satu realisasi inovasi penulis yang melihat jauh ke masa depan. Segala sesuatu yang kompleks dimulai dari sesuatu yang sederhana. Algoritma *greedy* termasuk algoritma paling sederhana dan salah satu yang paling tua. Kelebihan dari algoritma ini adalah algoritma ini mampu menyelesaikan atau menghampiri masalah optimisasi yang kompleks menjadi lebih sederhana dan efisien. Salah satunya adalah masalah *Autonomous Vehicle* ini. Dengan algoritma *greedy* sederhana dapat memecahkan masalah yang cukup kompleks.

V. Ucapan Terima Kasih

Pertama dan yang paling utama, penulis bersyukur kepada Tuhan Yang Maha Esa atas segala nikmat yang telah diberikan sehingga penulis dapat menyelesaikan makalah ini. Penulis juga berterima kasih kepada dosen mata kuliah Matematika Diskrit, bapak Judhi Santoso, ibu Harlili, dan khususnya bapak Rinaldi Munir yang telah membagikan ilmunya sehingga saya memiliki ilmu yang cukup untuk menyelesaikan ini. Terakhir, penulis berterima kasih kepada orang tua penulis yang terus mendukung dan memfasilitasi seluruh kebutuhan penulis.

References

[1] Levitin, Anany. 2011. *Introduction to The Design and Analysis of Algorithms.* London: Pearson.

[2] A. Silberchatz, P.B. Galvin, and G. Gagne. 2013. *Operating System Concepts 9th Edition.* New Jersey: John Wiley & Sons, Inc.

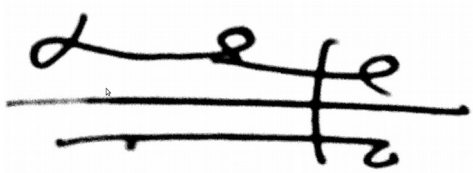
[3] Munir, Rinaldi. Diktat Kuliah IF2211 Strategi Algoritma. Program Studi Teknik Informatika ITB. 2018

[4] http://www.cs.cmu.edu/afs/cs/project/alv/www/index.html. *Navlab: The Carneige Mellon University Navigation Laboratory.* The Robotics Institute. Diakses pada tanggal 25 April 2019 21:47 GMT+7.

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, 29 April 2019



Azhar Abdurrasyid 13517097