JIRK
Journal of Innovation Research and Knowledge
Vol.3, No.5, Oktober 2023

SIMULASI PADA SISTEM OPERASI UNTUK MASALAH DINING PHILOSHOPERS

Oleh
Wiji Sariningsih
Program Studi Pendidikan Matematika,FMIPA, Universitas Indraprasta PGRI Jakarta
E-mail: wijisariningsih@gmail.com

Article History:
Received: 20-09-2023
Revised: 27-09-2023
Accepted: 23-10-2023

Keywords:
Sistem Operasi, Masalah Dining Philoshopers

Abstract: *Pada tahun 1965, Dijkstra mengemukakan dan memecahkan suatu masalah sinkronisasi yang disebut dengan Dining Philosophers Problem. Algoritma tersebut digunakan untuk mengatasi masalah kondisi bersaing pada penggunaan sumber yang terbatas oleh banyak pengguna. Program ini menggunakan metode semaphores untuk menghindari starvation dan deadlock. Untuk membantu menggambarkan perilaku sistem dan menganalisa hasil akhir jika diberi variasi input variabel waktu tunggu yang berbeda-beda serta untuk mempermudah dalam memahami dan mempelajari algoritma tersebut digunakan simulasi pendekatan pemecahan suatu masalah*

PENDAHULUAN

1. Latar Belakang

Studi dan aplikasi simulasi terus berkembang dan saat ini telah mencakup sangat banyak bidang dari berbagai disiplin ilmu, antara lain: ekonomi, biologi, fisika, rekayasa dan bahkan lingkungan. Tidak berlebihan jika dikatakan bahwa simulasi sepertinya sudah menjadi sesuatu yang harus. Studi simulasi ini menggambarkan sangat meluasnya penerapan komputer digital. Seperti masalah pada sistem operasi, dengan simulasi akan memudahkan dalam penganalisaan masalah tersebut. Konsep utama dalam sistem operasi adalah proses. Proses pada saat kejadian dapat merupakan suatu kegiatan yang bergantian dimana seolah-olah proses bekerja secara bersamaan. Hal ini dapat terjadi pada *multiprogramming*, yaitu beberapa program bisa berjalan secara bersamaan.

Pada masalah *Dining Philosophers*, bahwa sejumlah n filsuf duduk mengelilingi meja bundar yang didepannya ada sepiring spageti, dan secara bergantian berfikir dan makan spageti. Makan spageti dikehendaki oleh setiap filsuf. Dikanan dan kirinya hanya ada sebuah garpu. Sedangkan filsuf dapat makan jika bisa memegang kedua garpu kanan dan kirinya. Dengan adanya kejadian tersebut maka akan ada filsuf yang tidak bisa makan. Untuk melihat pemecahan masalah ini dan sekaligus mengetahui aplikasinya dalam sistem operasi, maka filsuf-filsuf digambarkan sebagai terminal-terminal yang saling berhubungan dalam sebuah jaringan komputer. Sedang proses makan dan berfikir bisa diasumsikan sebagai membaca dan menulis data ke disk, sedang garpu bisa diasumsikan sebagai sumber (source) yang bisa diakses oleh pengguna. Untuk menangani masalah tersebut akan digunakan metode semaphores. Semaphores adalah pendekatan yang dikemukakan oleh *Dijkstra* [DIJ-65]. Prinsip semaphores adalah sebagai berikut, dua proses atau lebih dapat bekerja sama dengan menggunakan penanda-penanda sederhana. Dalam metode

semaphores terdapat dua operasi yaitu *'UP* (menaikkan nilai *semaphores*) and *DOWN* (menurunkan nilai *semaphores*)'.

METODE PENELITIAN

Dalam penelitian ini peneliti menggunakan metode penelitian kuantitatif, tujuan dari penelitian eksperimen adalah untuk memperoleh informasi yang dapat diperoleh dari eksperimen yang sebenarnya dalam keadaan yang tidak memungkinkan untuk mengontrol semua variabel. Dalam analisa simulasi harus dilakukan spesifikasi prosedur secara lengkap dalam menggunakan dan merancang model simulasi, sehingga diperoleh aplikasi yang berhasil. Langkah-langkah dalam analisa simulasi antara lain perumusan masalah, pengumpulan dan analisa data, pengembangan model, validasi dan verifikasi, percobaan dan optimasi model, dan yang terakhir adalah implementasi.

HASIL DAN PEMBAHASAN

Algoritma Masalah Dining Philosophers.

Dalam teori perancangan sistem operasi terdapat bagian yang membahas tentang komunikasi antar proses. Pembahasan ini memang penting, khususnya pada sistem operasi yang bisa melakukan tugas majemuk (multitasking), yaitu beberapa proses bisa berjalan bersama-sama.

Pada tahun 1965, Djikstra mengajukan sebuah masalah sinkronisasi yang disebut dengan Dining Philosophers Problem. Masalah tersebut memang awalnya dari suatu cerita dengan beberapa filsuf. Masalah tersebut diceritakan sebagai berikut, lima filsuf duduk mengelilingi sebuah meja bundar. Setiap filsuf mempunyai sepiring spageti, untuk bisa makan spageti filsuf membutuhkan dua Ketika garpu. Sedangkan diantara piring tersebut hanya ada sebuah garpu. seorang filsuf merasa lapar, dia mencoba mengambil garpunya di kiri dan kanannya sekaligus. Jika berhasil mengambil kedua garpu tersebut dia akan makan sebentar, kemudian meletakkan garpu tersebut dan meneruskan berfikir demikian seterusnya. Masalahnya adalah bagaimana agar kelima filsuf tersebut bisa menjalani setiap prosesnya (makan dan berfikir) tanpa terganggu karena tidak berhasil mengambil garpu salah satu atau kedua-duanya sebab sedang digunakan oleh filsuf disampingnya.

Untuk melihat perancangan sistem operasi ini dan sekaligus mengetahui aplikasinya dalam teori perancangan sistem operasi, maka dalam program simulasi sistem operasi filsuf-filsuf digambarkan sebagai terminal- terminal yang saling berhubungan dalam sebuah jaringan komputer. Sedang proses makan dan berfikir bisa diasumsikan sebagai proses membaca dan menulis data ke disk, sedang garpu bisa sebagai sumber (source) yang bisa diakses oleh pengguna.

Sehingga manakala sebuah kejadian beberapa proses akan mengakses sebuah sumber yang terbatas secara bersama, maka apabila tanpa ada pengolahan komunikasi antar proses akan bisa menyebabkan kemacetan akibat kondisi diatas. Asumsi lengkapnya adalah bahwa algoritma ini digunakan untuk memecahkan masalah penggunaan sumber yang terbatas oleh banyak pengguna.

Gambar 3.1. Ilustrasi Filsuf mengelilingi meja makan.

Dalam permasalahan ini tidak ada dua filsuf berurutan yang dalam keadaan makan semua. Ada beberapa pendekatan yang bisa dipakai untuk memecahkan masalah ini. Pertama jika sebuah kejadian dimana seorang filsuf merasa lapar maka akan berusaha mengambil garpu. Jika tidak berhasil masuk proses makan maka ia akan menunggu sampai filsuf dikanan dan kirinya selesai proses makan. Ternyata solusi tersebut tidak dapat digunakan apabila kelima filsuf mengambil garpu kiri dan kanan secara bersamaan. Tidak seorang pun akan berhasil mengambil garpu dikiri atau kanannya, sehingga terjadi *deadlock*.

Cara pemecahan diatas dapat dimodifikasi lagi dengan cara setelah mengambil garpu kiri, program memeriksa kalau garpu kanan tersedia. Jika tidak tersedia, filsuf meletakkan garpu kirinya, menunggu beberapa saat dan lalu mengulangi keseluruhan proses. Tetapi usulan ini ternyata masih ada kemungkinan gagal juga. Dengan sedikit nasib buruk, semua filsuf dapat memulai dengan bersamaan, mengambil garpu kiri, melihat garpu kanan apakah tidak sedang digunakan, meletakkan garpu kirinya, lalu menunggu dan mengambil garpu kirinya kembali secara bersamaan, demikian seterusnya. Situasi semacam ini dimana semua program berjalan terus menerus tetapi selalu gagal kemudian masuk starvation.

Walaupun jika diperhatikan kemungkinan kejadian gagal diatas sangatlah kecil, meski demikian hal ini tetap tidak bisa dibenarkan, karena dalam beberapa aplikasi mutlak diperlukan sebuah pemecahan yang selalu bisa berjalan dan tidak ada kemungkinan gagal. Sebagai contoh adalah kontrol keamanan dalam reaktor nuklir, tentu harus bisa dipastikan tidak ada kejadian dimana sebuah program tidak mengalami macet ditengah proses.

Alternatif pemecahan selanjutnya yang tidak memiliki deadlock dan tidak ada kelaparan adalah pada saat seorang filsuf akan berusaha untuk makan maka ia akan melakukan penguncian semua filsuf menjadi dalam keadaan berfikir. Dari sudut pandang teoritis, pemecahan ini cukup memadai. Tetapi dari sudut pandang praktis pemecahan ini masih memiliki kesalahan (BUG) karena dengan pemecahan tersebut berarti hanya satu filsuf dapat makan pada setiap kesempatan.

Padahal jika ada enam filsuf maka seharusnya paling tidak ada tiga filsuf yang bisa makan secara bersama atau jika n adalah banyaknya filsuf yang aktif maka banyak filsuf yang bisa makan pada saat bersamaan adalah:

n div 2, dengan n = 2,3,4,5,....

Selanjutnya alternatif pemecahan terakhir berikut ini memungkinkan adanya paralelisme maksimum proses filsuf-filsuf yang aktif. Pemecahan yang paling baik yang memungkinkan adanya paralelisme maksimum proses filsuf- filsuf yang aktif adalah pemecahan dengan menggunakan sebuah barisan semaphores untuk keadaan seseorang

filsuf apakah sedang makan atau tidak. Pada pemecahan ini seorang filsuf hanya akan melangkah pada kondisi makan jika tetangganya (filsuf disebelah kiri dan kanannya) tidak makan.

Program ini menggunakan serangkaian semaphores. Satu semaphores untuk setiap filsuf, sehingga filsuf-filsuf yang sedang lapar dapat memblok jika garpu-garpu yang diperlukan sedang sibuk. Jadi dalam pemrograman ini seorang filsuf akan selalu mengecek keadaan/bit semaphores filsuf di sebelah kiri dan kanannya terlebih dahulu jika akan mencoba makan. Jika bit semaphores filsuf disebelahnya menunjukkan sedang proses makan maka filsuf akan berada dalam keadaan lapar dan akan mencoba lagi pada selang waktu dimana filsuf yang gagal masuk proses makan bisa mencoba lagi untuk masuk proses makan, bisa juga disebut waktu tunggu saat gagal makan.

Secara umum algoritma masalah dining philosophers (*Dining Philosophers Problem*) adalah sebagai berikut:

```
1. if phil_hungry i (aktif)
2.    then if phil i-1.(aktif)  ≠ eating
3.       then if phil i+1. (aktif)  ≠ eating
4.          then if aktif.dt3 Mod klok_kuant    ←   0
5.                then proses_makan
6.                else proses_lapar
7.          else proses_lapar
8.     else proses_lapar
9.   else if phil_eating (aktif)
10.       then kerja selesai, proses_berfikir
11. else proses_berfikir
12. if aktif   ←  awal antri
13. then k← k+1
```

Langkah dari algoritma DPP adalah sebagai berikut:

Langkah 1: Cek apakah filsuf i sedang dalam keadaan lapar yaitu dengan memanggil fungsi hungry. Jika benar maka lanjutkan ke langkah 3, jika salah maka lanjutkan ke langkah 9.

Langkah 2: Cek apakah filsuf i-1 tidak sedang dalam keadaan makan. Jika benar maka lanjutkan ke langkah 3, jika salah maka lanjutkan ke langkah 8.

Langkah 3: Cek apakah filsuf i+1 tidak sedang dalam keadaan makan. Jika pengecekan benar maka lanjutkan ke langkah 4, jika salah maka lanjutkan ke langkah7.

Langkah 4: Cek apakah aktif^.dt3 Mod lama waktu tunggu = 0, jika benar maka ke langkah 5. Jika salah lanjutkan ke langkah 6.

Langkah 5: Filsuf i dalam keadaan makan dengan memanggil prosedur
Proses Makan

Langkah 6: Filsuf i dalam keadaan lapar

Langkah 7: Filsuf i dalam keadaan lapar

Langkah 8: Filsuf i dalam keadaan lapar

Langkah 9: Cek apakah filsuf i makan, jika benar lanjutkan ke langkah 10, jika salah masuk ke langkah 11.

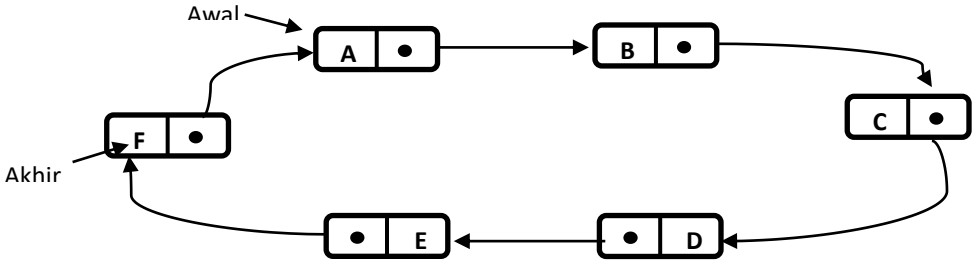
 ISSN 2798-3471 (Cetak)
ISSN 2798-3641 (Online)

Langkah 10: Setelah dalam keadaan lapar dan berusaha untuk makan dan terpenuhi maka langsung masuk proses berfikir.
Langkah 11: Filsuf i dalam keadaan berfikir, dengan memanggil prosedur Proses Berfikir.
Langkah 12: Cek aktif sama dengan awal antri, maka k=k+1 sehingga terpenuhi bahwa k mod kuantum =0.

Langkah ke 2 dan ke 3 adalah mengecek keadaan filsuf kanan dan kirinya, yaitu sebagai semaphores untuk memecahkan masalah ini agar terhindar dari *deadlock* dan *starvation.*

Struktur Data Program.

Masalah utama dalam program ini adalah menganalisa dan mengimplementasikan data berbentuk antrian. Pengertian antrian adalah suatu kumpulan data dimana penambahan elemen hanya bisa dilakukan pada salah satu ujung (disebut juga sisi belakang atau rear) dan penghapusan dilakukan lewat ujung lainnya (disebut sisi depan atau front).
Dalam program ini akan digunakan struktur data yang bertipe pointer dengan untai melingkar (circular link list). Dengan struktur data bertipe pointer, sebuah data berbentuk antrian bisa digambarkan sebagai berikut:



Gambar 3.2. Bentuk elemen antrian melingkar (*Circular*).

Dalam program struktur berisi elemen antrian :

Nama_job	Dt1	Dt2	Dt3	●

Keterangan:
Nama Job : berisi data bertipe karakter.
Dtl,Dt2,Dt3 : berisi data bertipe integer.

Struktur data elemen antrian tersebut digunakan dalam algoritma masalah dining filsufhers ini, dan elemen antrian akan berisi data filsuf. Elemen selengkapnya adalah seperti dalam tabel berikut:

Tabel 3.1. Isi field-field struktur data elemen antrian.

Nama Field	Dining Philosophers
Nama Job	Nama Filsuf
Dtl	Lama waktu makan
Dt2	Lama waktu berfikir
Dt3	Lama waktu lapar

Selain untuk membentuk data antrian data bertipe untai dalam programini juga digunakan untuk menyimpan data hasil proses. Data hasil proses algoritma berisi data-data seperti pada tabel berikut ini:

Tabel 3.2. Isi field-field struktur data hasil proses

Nama tipe data	Nama Field	Dining Philosophers
Node_Job_DPP	Nama Job	Nama Filsuf
	Dtl	Lama waktu makan
	Dt2	Lama waktu berfikir
	Dt3	Lama waktu lapar

Data bertipe dinamik diatas diperlukan karena banyaknya kerja yang diproses selama program berjalan tidak bisa ditentukan dari awal. Ini berbeda misalnya jika menggunakan tipe data larik dimana besarnya data sudah bisa ditentukan sebelumnya. Dengan tipe data dinamik ini pula maka penggunaan memori bisa menjadi lebih efisien karena dengan data dinamik tersebut maka lokasi memori yang sebelumnya digunakan untuk menyimpan data setelah data tidak digunakan lagi maka lokasi memori tersebut bisa dibebaskan atau dikosongkan lagi.

Deklarasi Tipe Data.

Setelah menganalisa dan menentukan struktur dan bentuk untai antrian yang dipakai dalam program maka langkah selanjutnya adalah mendeklarasikan dalam bahasa yang digunakan yaitu bahasa Pascal, tipe data pointer juga dideklarasikan pada bagian deklarasi tipe implementasi untuk untai antrian adalah sebagai berikut:

```
Type Antri      =^ Node_Job;
     Node_Job = Record;
Nama_Job: Char;
Dtl,Dt2,Dt3 : Integer;
Berikut : Antri;
End;
```

Keterangan:

Antri : Nama variabel yang bertipe pointer.

Node_Job : Nama simpul.

Nama_Job, Dt1, Dt2, Dt3 : Nama field yang merupakan isi record Node_Job.

Berikut : Nama field yang bertipe data pointer (akan menunjukkan halaman simpul berikutnya).

Sedang untuk untai data hasil proses berisi empat field data implementasi yaitu:

```
Dbase hsl = ^Data_Job_DPP;
Data_Job_DPP = Record
Nama Job:Char;
Dt1,Dt2,Dt3 :Integer;
Berikut : Dbase_hsl;
End;
```

Analisa Model Simulasi.

Program simulasi ini selain menggambarkan perintah sistem juga bisa untuk mengetahui unjuk kerja sistem dengan cara mengamati output program pada berbagai variasi input. Model simulasi yang dihasilkan oleh program ini bisa dilihat dari beberapa sudut pandang. Jika dilihat dari tampilan program simulasi yang menggunakan teknik animasi grafis maka bisa dikatakan bahwa model simulasi ini bisa diklasifikasikan kedalam model simulasi deskriptif, karena yang nampak hanya penggambaran perilaku sistem, tetapi jika dilihat dari statistik output yang dikeluarkan program ini maka model simulasi ini juga bisa diklasifikasikan ke dalam model simulasi preskriptif, karena statistik output tersebut digunakan untuk mengamati unjuk kerja/optimalisasi simulasi algoritma yang dijalankan, selanjutnya model simulasi yang dihasilkan program ini bisa digolongkan sebagai model simulasi deterministik apabila input data dilakukan secara manual.

Output Masalah Dining Philosophers.

Tampilan simulasi dalam pembahasan ini lebih condong ke dalam model simulasi deskriptif. Tampilan mengilustrasikan periode yang sedang dijalani oleh setiap filsuf yang sedang aktif. Ilustrasi sebagai berikut:

1. Jika pada gambar ilustrasi filsuf muncul icon dengan pesan 'EAT', maka filsuf sedang berada pada periode makan.

Gambar 3.3. Ilustrasi filsuf makan.

2. Jika pada gambar ilustrasi filsuf muncul icon dengan pesan 'THINK', maka filsuf sedang berada pada periode berfikir.

Gambar 3.4. Ilustrasi filsuf berfikir.

3. Jika filsuf berada dalam keadaan lapar gambar muncul icon dengan pesan

Gambar 3.5. Ilustrasi filsuf lapar.

Meskipun simulasi pada sistem operasi ini termasuk dalam model deskriptif tetapi statistik output program ini juga digunakan untuk menganalisis output program jika diberikan variasi input. Sebagai contoh untuk dibahas, bagaimana output program jika nilai waktu tunggu saat gagal makan diberikan besar atau sebaliknya.

Berdasarkan teori, jika sebuah kejadian dimana filsuf mencoba masuk proses makan dan gagal makan maka akan mendapat waktu tunggu sampai mendapat kesempatan untuk mencoba masuk proses makan. Dalam hal ini pemberian nilai waktu tunggu saat gagal masuk proses makan mempengaruhi rata-rata waktu berfikir dan makan bagi setiap filsuf. Di bawah ini akan diberikan contohnya dan untuk mengetahui tingkah laku filsuf tiap-tiap klok_proses .

Contoh 1. Apabila diberi masukan data dengan banyak filsuf 2, periode lama proses 20 dan lama waktu tunggu untuk bisa makan sebesar 3 dan 10 maka akan dihasilkan output seperti pada tabel 3.3 dan tabel 3.4 sebagai berikut:

Tabel 3.3. Hasil komputasi banyak filsuf 2, lama proses 20, waktu tunggu 3
Output

Philo shoper	Think			Eat			Hungry		
	Cacah	Lama	Rata	Cacah	Lama	Rata	Cacah	Lama	Rata
A	1	5	5.00	2	6	3.00	1	9	9.00
B	3	8	2.67	2	7	3.50	1	5	5.00
Jumlah	4	13		4	13		2	14	

Banyak Philosop yang aktif = 2
Periode lama proses = 20
Waktu tunggu = 3

Tabel 3.3. Hasil komputasi banyak filsuf 2, lama proses 20, waktu tunggu 10
Output

Philo shoper	Think			Eat			Hungry		
	Cacah	Lama	Rata	Cacah	Lama	Rata	Cacah	Lama	Rata
A	1	3	3.00	0	0	0.00	1	17	17.00
B	4	9	2.25	3	11	3.67	0	0	0.00
Jumlah	5	12		3	11		1	17	

Banyak Philosop yang aktif = 2
Periode lama proses = 20
Waktu tunggu = 10

Contoh 2: Apabila diberi masukan data dengan banyak filsuf 3, periode lama proses 60 dan lama waktu tunggu untuk bisa makan sebesar 10 dan 40 maka akan dihasilkan output seperti pada tabel 3.5 dan tabel 3.6 sebagai berikut:

Tabel 3.5. Hasil komputasi banyak filsuf 3, lama proses 60 waktu tunggu 10.
Output

Philo shoper	Think			Eat			Hungry		
	Cacah	Lama	Rata	Cacah	Lama	Rata	Cacah	Lama	Rata
A	4	6	1.50	3	6	2.00	2	48	24.00
B	5	15	3.00	5	15	3.00	1	30	30.00
C	5	17	3.40	5	13	2.60	2	30	15.00
Jumlah	13	38		13	34		5	108	

Banyak Philosop yang aktif = 3
Periode lama proses = 60
Waktu tunggu = 10

ISSN 2798-3471 (Cetak)
ISSN 2798-3641 (Online)

Tabel 3.6. Hasil komputasi banyak filsuf 3, lama proses 60, waktu tunggu 40.
Output

Philo shoper	Think			Eat			Hungry		
	Cacah	Lama	Rata	Cacah	Lama	Rata	Cacah	Lama	Rata
A	5	9	1.80	6	11	1.83	1	40	40.00
B	0	0	0	0	0	0	1	60	60.00
C	8	22	2.75	7	22	3.14	1	16	16.00
Jumlah	13	31		13	33		3	116	

Banyak Philosop yang aktif = 3
Periode lama proses = 60
Waktu tunggu = 40

Contoh 3. Apabila diberi masukan data dengan banyak filsuf 4, periode lama proses 30 dan lama waktu tunggu untuk bisa makan sebesar 5 dan 20 maka akan dihasilkan output seperti [pada tabel 3.7 dan tabel 3.8 sebagai berikut.

Tabel 3.7. Hasil komputasi banyak filsuf 4, lama proses 30, waktu tunggu 5.
Output

Philo shoper	Think			Eat			Hungry		
	Cacah	Lama	Rata	Cacah	Lama	Rata	Cacah	Lama	Rata
A	4	11	2.75	3	7	2.33	2	12	6.00
B	3	6	2.00	3	9	3.00	1	15	15.00
C	2	8	4.00	2	9	4.50	2	13	6.50
D	3	6	2.00	3	9	3.00	1	15	15.00
Jumlah	12	31		11	25		6	55	

Banyak Philosop yang aktif = 4
Periode lama proses = 30
Waktu tunggu = 5

Tabel 3.8. Hasil komputasi banyak filsuf 4, lama proses 30, waktu tunggu 20.
Output

Philo shoper	Think			Eat			Hungry		
	Cacah	Lama	Rata	Cacah	Lama	Rata	Cacah	Lama	Rata
A	2	3	1.50	1	7	7.00	1	20	20.00
B	5	12	2.40	4	10	2.50	1	8	8.00
C	2	3	1.50	1	7	7.00	1	20	20.00
D	5	9	1.80	5	12	2.40	1	9	9.00
Jumlah	14	27		11	36		4	57	

Banyak Philosop yang aktif = 4
Periode lama proses = 30
Waktu tunggu = 20

Contoh 4. Apabila diberi masukan data dengan banyak filsuf 5, periode lama proses 80 dan lama waktu tunggu untuk bisa makan sebesar 30 dan 40 maka akan dihasilkan output seperti pada tabel 3.9 dan tabel 3.10 sebagai berikut:

Tabel 3.9. Hasil komputasi banyak filsuf 5, lama proses 80, waktu tunggu 30.

Output

Philo shoper	Think			Eat			Hungry		
	Cacah	Lama	Rata	Cacah	Lama	Rata	Cacah	Lama	Rata
A	9	19	2.11	8	15	1.88	1	46	46.00
B	8	16	2.00	7	19	2.71	2	45	22.50
C	6	11	1.83	6	9	1.50	1	60	60.00
D	9	18	2.00	8	16	2.00	1	46	46.00
E	15	21	1.40	15	29	1.93	1	30	30.00
Jumlah	47	85		44	88		6	227	

Banyak Philosop yang aktif = 5
Periode lama proses = 80
Waktu tunggu = 30

Tabel 3.10: Hasil komputasi banyak filsuf 5, lama proses 80, waktu tunggu 40.

Output

Philo shoper	Think			Eat			Hungry		
	Cacah	Lama	Rata	Cacah	Lama	Rata	Cacah	Lama	Rata
A	11	21	1.91	11	20	1.82	1	39	39.00
B	10	17	1.70	11	23	2.09	1	40	40.00
C	12	20	1.67	12	21	1.75	1	39	39.00
D	8	15	1.88	8	25	3.13	1	40	40.00
E	0	0	0	0	0	0	1	80	80.00
Jumlah	41	73		42	89		5	228	

Banyak Philosop yang aktif = 5
Periode lama proses = 80
Waktu tunggu = 40

Dari hasil diatas dapat dilihat bahwa pemberian waktu tunggu saat gagal makan yang terlalu tinggi akan meningkatkan rata-rata waktu lapar sehingga menyebabkan sebagian besar waktu pilosop berada pada periode lapar. Pemberian nilai waktu tunggu saat gagal makan yang terlalu tinggi mengakibatkan rata-rata waktu berfikir dan waktu makan setiap filsuf kecil. Dalam program ini dapat dilihat bahwa penentuan nilai diatas tergantung banyaknya filsuf yang aktif (2-5 yang aktif).

KESIMPULAN

Dari uraian yang telah dibahas sebelumnya maka dalam tugas akhir ini dapat disimpulkan bahwa dengan menggunakan teori analisa simulasi maka akan mempermudah dalam pemahaman permasalahan yang ada. Seperti algoritma Dining Philosophers Problem

dalam sistem operasi ini, dapat diamati tingkah laku dari filsuf (sedang makan, berfikir atau lapar) dengan banyak filsuf minimum 2 dan maksimum 5 dan lama proses maksimum adalah 99 satuan waktu. Bahwa dengan pemberian variabel masukan waktu tunggu yang terlalu tinggi pada filsuf akan mengakibatkan sebagian besar waktu filsuf berada dalam periode lapar. Sedangkan dengan waktu tunggu filsuf yang kecil akan mengakibatkan lama lapar filsuf juga kecil dan filsuf akan sering melakukan aktivitas makan dan berfikir. Dalam simulasi ini variabel masukan waktu tunggu adalah lebih kecil atau sama dengan lama proses.

DAFTAR PUSTAKA

[1] Bambang Hariyanto, Ir, *Sistem Operasi*, Edisi Kedua, Informatika, Bandung, 1995.
[2] Deitel, H.M, *The Systems Programming Series an Introduction to Operating Systems,* Addison Wesley, 1982.
[3] Hoover, Stewart V and Perry, Ronald, F, *Simulation a Problem Solving Approach,* Adisson Wesley Publishing, 1998.
[4] Insap Santosa, P, Ir, M. Sc*, Struktur Data Menggunakan Turbo Pascal 6.0*, Andi Offset, 1993.
[5] Jogiyanto, H.M, *Turbo Pascal Jilid 1 dan Jilid 2*, Andi Offset, 1998.
[6] Laksono, Yoyok Adisetio, *Membuat Program Grafik Lewat Turbo Pascal 5.5/6.0,* Edisi Kedua, Andi Offset, 1995.
[7] Mokarribin A Rahman, *Dining Philosophers Problem*, CSC 413 Operating System, 2002.
[8] Sri Kusumadewi, *Sistem Operasi*, Edisi Pertama, J&J Learning, 2000.
[9] Tanenbaum, Andrew S, *Operation Systems Design and Implementation*, Prentice Hall, 1987.

HALAMAN INI SENGAJA DIKOSONGKAN

ISSN 2798-3471 (Cetak)
ISSN 2798-3641 (Online)