PERANGKAT LUNAK SIMULASI DEADLOCK MENGGUNAKAN ILUSTRASI DINING PHILOSOPHERS PROBLEM

Arfiani Nur Khusna[1)], Nur Rochmah Dyah PA[2)]
[1,2)]Teknik Informatika, Universitas Ahmad Dahlan Yogyakarta
[1,2)]Jl. Prof. Dr. Soepomo, S.H., Janturan, Warungboto, Umbulharjo, Yogyakarta 55164
e-mail: arfiani.khusna@tif.uad.ac.id[1)], rochmahdyah@tif.uad.ac.id[2)]

ABSTRAK

Sistem operasi harus mampu melakukan pengontrolan penggunaan resource. Dalam proses perancangan sistem operasi, terdapat suatu landasan umum yang disebut dengan kongkurensi. Proses-proses disebut kongkuren jika proses-proses (lebih dari satu proses) berada pada waktu yang bersamaan. Keadaan ini disebut dengan multitasking dari sistem operasi. Proses-proses kongkuren dapat sepenuhnya tak bergantung dengan lainnya tapi dapat juga saling berinteraksi. Proses-proses yang berinteraksi memerlukan sinkronisasi agar terkendali dengan baik. Namun, pada proses-proses kongkuren yang berinteraksi, terdapat masalah yang harus diselesaikan seperti deadlock dan sinkronisasi. Deadlock terjadi karena sekumpulan proses-proses yang di blok dimana setiap proses membawa sebuah sumber daya dan menunggu mendapatkan sumber daya yang dibawa oleh proses lain sehingga terdapat keadaan menunggu yang tidak akan pernah berakhir (kebuntuan).

Dining Philosophers Problem merupakan ilustrasi kemungkinan terjadinya deadlock, yaitu suatu kondisi dimana dua proses atau lebih tidak dapat meneruskan eksekusinya karena saling menunggu untuk menggunakan sumber daya. Dining Philosophers Problem dapat diilustrasikan terdapat lima orang filsuf yang sedang duduk mengelilingi sebuah meja. Lima orang filsuf merupakan ilustrasi dari lima proses dengan keadaan proses yang saling menunggu untuk menggunakan sumber daya.

Hasil penelitian ini berupa perangkat lunak untuk menggambarkan keadaan deadlock menggunakan ilustrasi dining philosophers problem dengan ilustrasi semua filsuf sedang berada dalam kondisi lapar dan memegang sumpit di tangan sebelah kiri, maka akan terjadi kondisi deadlock.

***Kata Kunci**: deadlock, dining philosophers problem, simulasi*

ABSTRACT

The operating system must be able to control resource usage. In the process of designing the operating system, there is a common foundation called kongkurensi. Processes called kongkuren if processes (more than one process) are at the same time. This is called the multitasking operating system. Kongkuren processes can be completely independent of the other but can also interact with each other. Processes that require synchronization in order to interact properly controlled. However, the processes that interact kongkuren, there are problems to be solved such as deadlock and synchronization. Deadlock occurs because a set of processes in a block where every process of bringing a resource and waiting to get the resources brought by another process so there is a wait state which will never expire (deadlock).

Dining Philosophers Problem is an illustration of the possibility of a deadlock, a condition in which two or more processes can not continue its execution as they waited for the use of resources. Dining Philosophers problem can be illustrated, there are five philosophers sitting around a table. Five philosophers is an illustration of the five processes with the state of the process of mutual waiting to use the resources.

The results of this study are to describe the state of software deadlocks using illustrations dining philosophers problem with illustrations of all the philosophers were in a state of hunger and holding chopsticks in hand to the left, there will be a deadlock condition.

***Keywords**: deadlock, dining philosophers problem, simulation*

I. PENDAHULUAN

Sistem operasi merupakan suatu program yang bertindak sebagai *interface* antara *user* dan sistem komputer. Sistem ope suatu landasan umum yang disebut dengan kongkurensi. Proses-proses itu disebut kongkuren jika proses-proses rasi ini harus mampu melakukan pengontrolan penggunaan *resource*. Dalam proses perancangan sistem operasi, terdapat (lebih dari satu proses) berada pada saat yang sama. Keadaan ini disebut dengan *multitasking* dari sistem operasi. Proses-proses kongkuren dapat sepenuhnya tak bergantung dengan lainnya tapi dapat juga saling berinteraksi. Proses-proses yang berinteraksi memerlukan sinkronisasi agar terkendali dengan baik. Namun, pada proses-proses kongkuren yang berinteraksi, terdapat beberapa masalah yang harus diselesaikan seperti *deadlock* dan sinkronisasi.

Permasalahan *deadlock* terjadi karena sekumpulan proses-proses yang di-blok dimana setiap proses membawa sebuah sumber daya dan menunggu mendapatkan sumber daya yang dibawa oleh proses lain, atau suatu proses menunggu suatu kejadian tertentu yang tidak akan pernah terjadi, karena kejadian itu hanya bisa dilakukan oleh

proses lain sehingga terdapat keadaan menunggu yang tidak akan pernah berakhir (kebuntuan). Salah satu masalah klasik yang dapat menggambarkan masalah tersebut adalah *Dining Philosophers Problem.*

Dining Philosophers Problem dapat diilustrasikan sebagai berikut, terdapat lima orang filsuf yang akan makan. Di meja disediakan lima buah sumpit. Jika filsuf lapar betul, maka ia akan mengambil dua buah sumpit, yaitu di tangan kanan dan kiri. Namun adakalanya hanya diambil sumpit satu saja. Jika ada filsuf yang mengambil dua buah sumpit, maka ada filsuf yang harus menunggu sampai sumpit tersebut ditaruh kembali. Di dalam problema ini ada kemungkinan terjadi *deadlock*, yaitu suatu kondisi dimana dua proses atau lebih tidak dapat meneruskan eksekusinya. [1]

Penelitian ini bertujuan untuk menyelesaikan masalah tersebut, dengan cara merancang suatu perangkat lunak yang dapat mensimulasikan proses kerja dari problem tersebut sekaligus mensimulasikan pencegahan masalah *deadlock* tersebut.

II. METODE

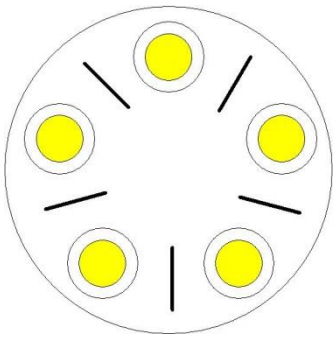
1. Analisis alur *Dining Philosophers Problem*

Mengidentifikasi asumsi dari *Dining Philosopher Problems*, dengan ilustrasi yaitu terdapat lima orang filsuf yang sedang duduk mengelilingi sebuah meja. Terdapat lima mangkuk mie di depan masing-masing filsuf dan satu sumpit di antara masing-masing filsuf. Para filsuf menghabiskan waktu dengan berpikir (ketika kenyang) dan makan (ketika lapar). Ketika lapar, filsuf akan mengambil dua buah sumpit (di tangan kiri dan tangan kanan) dan makan. Namun adakalanya, hanya diambil satu sumpit saja. Jika ada filsuf yang mengambil dua buah sumpit, maka dua filsuf di samping filsuf yang sedang makan harus menunggu sampai sumpit ditaruh kembali. Jika tiap-tiap filsuf lapar dan mengambil sumpit kiri, maka semua nilai sumpit=0, dan kemudian tiap-tiap filsuf akan mengambil sumpit kanan, maka akan terjadi *deadlock*. [2]

Hal ini dapat diimplementasikan dengan *wait* dan *signal* pada gambar 1, pada gambar 2 terdapat ilustrasi lima filsuf dalam satu meja makan.

```
repeat
      wait (sumpit[i]);
      wait (sumpit [i+1 mod 5]);
            ...
            makan
            ...
      signal (sumpit[i]);
      signal (sumpit[i+1 mod 5]);
            ...
            berpikir
            ...
until false
```

Gambar 1. Struktur proses fisuf ke-i



Gambar 2. Lima filsuf dalam satu meja makan

2. Implementasi

Implementasi perangkat lunak simulasi menggunakan bahasa pemrograman *Microsoft Visual Basic 6.0*. [5]

III. HASIL

A. Analisis alur *Dining Philosophers*

Dining Philosohers Problem dapat diilustrasikan sebagai berikut, terdapat lima orang filsuf yang sedang duduk mengelilingi sebuah meja. Terdapat lima mangkuk mie di depan masing-masing filsuf dan satu sumpit di antara masing-masing filsuf. Para filsuf menghabiskan waktu dengan berpikir (ketika kenyang) dan makan (ketika lapar). Ketika lapar, filsuf akan mengambil dua buah sumpit (di tangan kiri dan tangan kanan) dan makan. Namun adakalanya, hanya diambil satu sumpit saja. Jika ada filsuf yang mengambil dua buah sumpit, maka dua filsuf di samping filsuf yang sedang makan harus menunggu sampai sumpit ditaruh kembali. [3]

Dalam proses perancangan perangkat lunak simulasi ini, penulis mengambil beberapa asumsi, yaitu:

1. Hanya terdapat 5 (lima) orang filsuf dalam proses simulasi.
2. Masing-masing filsuf memiliki kondisi sebagai berikut:

a. Kenyang.

Filsuf akan berada dalam kondisi kenyang sesaat setelah makan.

b. Lapar.

Beberapa saat setelah makan, filsuf akan merasa lapar.

c. Mati.

Beberapa saat setelah merasa lapar dan apabila filsuf belum juga mendapatkan sumpit, maka filsuf akan mati.

3. Aksi yang dapat dilakukan oleh masing-masing filsuf, yaitu:

a. Berpikir.

Filsuf akan berpikir apabila filsuf berada dalam kondisi kenyang.

b. Cari Sumpit.

Filsuf akan mencari dan mengambil sumpit di kedua tangannya apabila filsuf berada dalam kondisi lapar.

c. Makan

Apabila filsuf mendapatkan dua sumpit di kedua tangannya, maka filsuf akan makan hingga kenyang. Setelah kenyang, filsuf kembali berpikir.

4. Di dalam perangkat lunak, kondisi filsuf di atas secara matematis menjadi waktu-A dan waktu-B.

a. Waktu-A, yaitu waktu yang diperlukan untuk mengubah kondisi filsuf dari kenyang menjadi lapar (dalam detik).

b. Waktu-B, yaitu waktu yang diperlukan untuk mengubah kondisi filsuf dari lapar menjadi mati (dalam detik).

c. Masa Hidup, yaitu masa hidup filsuf pada saat itu (dalam detik). Maksimum masa hidup adalah sebesar nilai waktu-A + waktu-B. Masa hidup filsuf akan bertambah ketika filsuf sedang makan dan akan senantiasa berkurang di luar aksi itu.

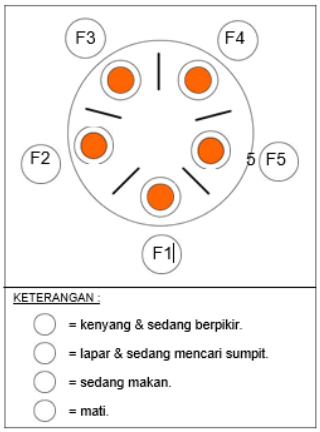
Ketiga komponen ini menjadi ukuran matematis kondisi filsuf di dalam perangkat lunak. Misalnya, seorang filsuf memiliki waktu-A = 10 detik, waktu-B = 8 detik dan masa hidup = 12 detik. Karena masa hidup filsuf adalah 12 detik atau lebih besar 4 detik daripada waktu-B, maka dapat dihitung bahwa 4 detik kemudian, filsuf akan mulai merasa lapar dan mulai mencari sumpit. Bila filsuf mendapatkan dua sumpit di kedua tangannya, maka filsuf akan mulai memakan mie yang ada di depannya. Ketika sedang makan, masa hidup filsuf akan bertambah kembali. Filsuf tidak akan berhenti makan hingga filsuf berada dalam kondisi kenyang atau masa hidupnya mencapai nilai maksimum yaitu 18 detik (waktu-A + waktu-B). Setelah itu, filsuf akan berpikir dalam 10 detik berikutnya, sebelum merasa lapar dan mencari sumpit lagi. Apabila filsuf merasa lapar dan tidak mendapatkan sumpit dalam waktu 8 detik (waktu-B), maka masa hidup filsuf akan menjadi nol dan filsuf akan mati. Untuk mendapatkan sumpit, filsuf harus menunggu apabila sumpit sedang dipakai oleh filsuf di sampingnya. Dalam kondisi itu, masa hidup filsuf akan terus berkurang. [4]

Untuk dapat lebih memahami proses yang terjadi, perhatikan contoh illustrasi berikut ini. Misalkan properti 5 orang filsuf dalam simulasi adalah sebagai berikut:

TABEL 1. TABEL FILSUF

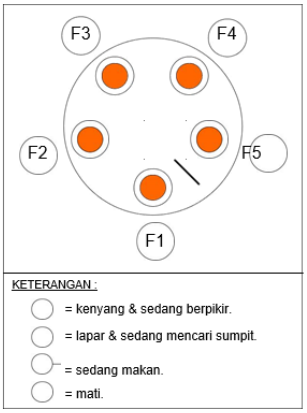
	Waktu-A (sekon)	Waktu-B (sekon)	Kondisi Awal (sekon)
Filsuf-1	7	10	15
Filsuf-2	5	4	4
Filsuf-3	6	11	14
Filsuf-4	5	13	11
Filsuf-5	6	12	18

Dari tabel dapat diketahui kondisi masing-masing filsuf, yaitu filsuf-1 berada dalam kondisi kenyang karena kondisi awal 15 sekon berada di atas waktu-B yang hanya 10 sekon (filsuf-1 akan merasa lapar dan mencari sumpit 5 sekon kemudian). Filsuf-2 berada dalam kondisi lapar karena kondisi awal 4 sekon = waktu-B, filsuf-3 berada dalam kondisi kenyang, filsuf-4 berada dalam kondisi lapar dan filsuf-5 berada dalam kondisi kenyang. Kondisi awal problema *dining philosophers* dapat digambarkan dengan bagan illustrasi sebagai berikut:



Gambar 3. Bagan illustrasi kondisi awal simulasi

Proses simulasi adalah sebagai berikut pada saat *t* = 1 sekon, filsuf-1, filsuf-3 dan filsuf-5 kenyang dan sedang berpikir, sedangkan filsuf-2 dan filsuf-4 lapar dan mendapatkan sumpit di tangan kiri. Pada saat *t* = 2 sekon, filsuf-2 dan filsuf-4 mendapatkan 2 sumpit dan mulai makan, sedangkan filsuf-1, filsuf-3 dan filsuf-5 masih kenyang dan sedang berpikir. Pada saat *t* = 3 sekon, filsuf-3 merasa lapar (karena masa hidup filsuf-3 saat ini = waktu-B yaitu 11 sekon) dan mulai mencari sumpit, tetapi tidak mendapatkan sumpit karena sumpit di sebelah kiri digunakan oleh filsuf-4 dan sumpit di sebelah kanan digunakan oleh filsuf-2. Bagan illustrasinya adalah sebagai berikut:



Gambar 4. Bagan illustrasi kondisi simulasi saat t=3 sekon

Pada saat *t* = 5 sekon, filsuf-1 merasa lapar (karena masa hidup filsuf-1 saat ini = waktu-B yaitu 10 sekon) dan mencari sumpit. Filsuf-1 mendapatkan sumpit di tangan kanan. Pada saat *t* = 6 sekon, filsuf-5 merasa lapar (karena masa hidup filsuf-5 saat ini = waktu-B yaitu 12 sekon) dan mencari sumpit. Filsuf-5 tidak mendapatkan sumpit. Pada saat *t* = 9 sekon, filsuf-2 kenyang (karena masa hidupnya sudah mencapai nilai maksimum, yaitu 9 sekon = waktu-A + waktu-B) dan mulai berpikir. Filsuf-3 mendapatkan sumpit di tangan kanan. Pada saat *t* = 10 sekon, filsuf-1 mendapatkan 2 sumpit dan mulai makan. Pada saat *t* = 11 sekon, filsuf-4 kenyang dan mulai berpikir. Filsuf-5 mendapatkan sumpit di tangan kanan. Pada saat *t* = 12 sekon, filsuf-3 mendapatkan 2 sumpit dan mulai makan.

Simulasi hanya akan berhenti apabila terjadi kondisi *deadlock*. Kondisi *deadlock* dalam simulasi *dining philosophers problem* terjadi apabila pada satu saat, semua filsuf merasa lapar secara bersamaan dan semua filsuf mengambil sumpit di tangan kiri. Pada saat filsuf akan mengambil sumpit di tangan kanan, maka terjadilah kondisi *deadlock*, karena semua filsuf akan saling menunggu sumpit di sebelah kanannya (kondisi yang tidak akan pernah terjadi). Untuk kasus *deadlock*, perhatikan kondisi berikut:

TABEL 2. TABEL FILSUF UNTUK KASUS *DEADLOCK*

	Waktu-A (sekon)	Waktu-B (sekon)	Kondisi Awal (sekon)
Filsuf-1	17	12	27
Filsuf-2	5	3	2
Filsuf-3	15	10	25
Filsuf-4	6	5	5
Filsuf-5	20	5	20

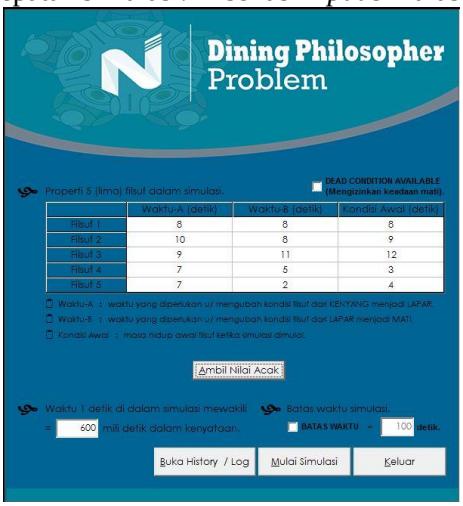
Pada saat *t* = 1 sekon, filsuf-2 dan filsuf-4 lapar dan mendapatkan sumpit di tangan kiri, sedangkan filsuf-1, filsuf-3 dan filsuf-5 kenyang dan sedang berpikir. Pada saat *t* = 2 sekon, filsuf-2 dan filsuf-4 mendapat 2 sumpit dan mulai makan. Pada saat *t* = 10 sekon, filsuf-2 dan filsuf-4 kenyang dan mulai berpikir. Pada saat *t* = 15 sekon, semua filsuf

secara bersamaan lapar dan mengambil sumpit di tangan kiri. Pada saat ini, telah terjadi kondisi *deadlock*, karena semua filsuf yang sedang mengenggam sumpit di tangan kiri menunggu sumpit di sebelah kanan. Semua filsuf akan saling menunggu.

B. Implementasi

1. *Input* simulasi

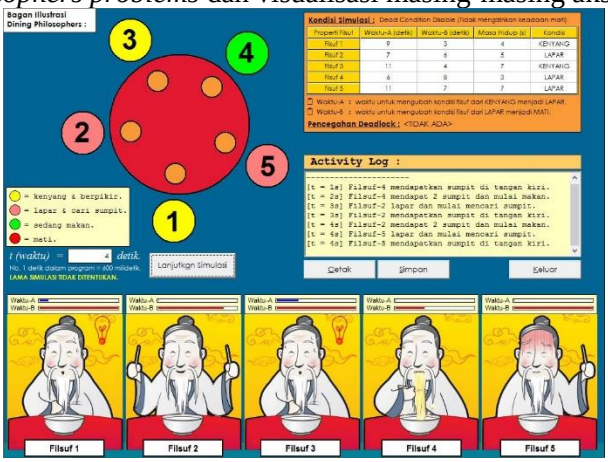
Input simulasi berfungsi untuk memasukkan nilai. Nilai simulasi dapat diambil dari nilai acak pada komputer. Tampilan *input* data pada simulasi berupa properti filsuf yang berisi waktu. Waktu ini berfungsi untuk mengubah kondisi kenyang menjadi lapar, lapar menjadi mati serta masa awal hidup filsuf. Peringatan *deadlock* digunakan ketika terjadi kondisi *deadlock* dimana semua proses akan berhenti. Opsi '*dead condition available*' diizinkan apabila salah satu filsuf didalam simulasi untuk tidak meneruskan prosesnya. Kecepatan simulasi berfungsi untuk mengatur lama waktu simulasi dan kecepatan simulasi. Eksekusi *input* simulasi akan dilanjutkan ke proses simulasi.



Gambar 5. Input simulasi

2. Simulasi Ketika Tidak Terjadi Deadlock

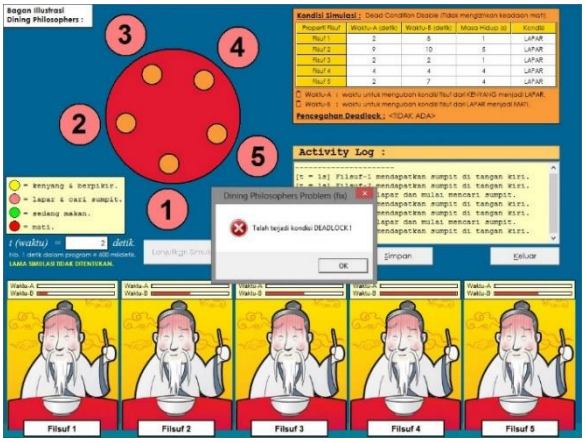
Proses simulasi berfungsi untuk mensimulasikan proses *dining philosophers problem*. Pada tampilan ini, akan ditampilkan bagan *dining philosophers problems* dan visualisasi masing-masing aksi yang dilakukan oleh 5 filsuf.



Gambar 6. Simulasi tidak terjadi deadlock

3. Simulasi Terjadi Deadlock

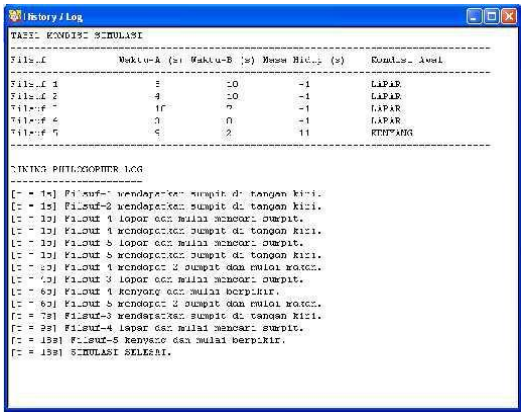
Simulasi terjadinya *deadlock* akan menampilkan nilai yang telah dimasukkan pada *input*. Tampilan berfungsi untuk mengetahui terjadinya *deadlock*. *Deadlock* terjadi karena pada tahap sebelumnya tidak diberikan solusi untuk pencegahannya, sehingga akan muncul tampilan sebagai berikut.



Gambar 7. Simulasi Terjadi Deadlock

4. History Activity Log

History Activity Log berfungsi untuk mencatat dan menampilkan laporan proses yang telah terjadi pada proses simulasi *Dining Philosopher Problems.*



Gambar 8. *History Activity Log*

IV. SIMPULAN DAN SARAN

A. Kesimpulan

Dihasilkan simulasi menggunakan *Dining Philosophers Problem* untuk menggambarkan proses terjadinya *deadlock* pada sistem operasi. Pada *Dining Philosophers Problem* apabila semua filsuf sedang berada dalam kondisi lapar dan memegang sumpit di tangan sebelah kiri, maka akan terjadi kondisi *deadlock.*

B. Saran

Saran yang dapat membantu dalam pengembangan perangkat lunak simulasi ini yaitu simulasi ditambahkan solusi untuk mencegah atau menghindari kondisi *deadlock* karena simulasi yang dihasilkan saat ini belum dapat mencegah *deadlock.*

REFERENSI

[1] Ali Pangera, Abas &Ariyus, Doni. 2010. Sistem Operasi. Yogyakarta: Andi Yogyakarta.
[2] B. Wibowo, dkk. 2003. *Sistem Operasi Bahan Kuliah IKI-20230 BAB I*, hal. 1-3.
[3] Hariyanto, Bambang. 2009, Revisi keempat. Sistem Operasi**,** Bandung: Informatika.
[4] Stalling, William, "Sistem Operasi : Internal Dan Prinsip-Prinsip Perancangan", PT Indeks Kelompok Gramedia, Jakarta, 2006.
[5] Wahana Komputer, "Pemrograman VB 6.0", Andi, Yogyakarta, 2000