SIMULASI PENCEGAHAN *DEADLOCK* MENGGUNAKAN *DINING PHILOSOPHERS PROBLEMS*

TUGAS AKHIR

Diajukan Sebagai Salah Satu Syarat
Untuk Memperoleh Gelar Sarjana Teknik pada
Jurusan Teknik Informatika

oleh :

THOIF CHUSAINI
10451025568

JURUSAN TEKNIK INFORMATIKA
FAKULTAS SAINS DAN TEKNOLOGI
UNIVERSITAS ISLAM NEGERI SULTAN SYARIF KASIM RIAU
PEKANBARU
2011

SIMULASI PENCEGAHAN *DEADLOCK* MENGGUNAKAN *DINING PHILOSOPHERS PROBLEMS*

THOIF CHUSAINI
NIM : 10451025568

Tanggal Sidang : 22 Juni 2011
Periode Wisuda : November 2011

Jurusan Teknik Informatika
Fakultas Sains dan Teknologi
Universitas Islam Negeri Sultan Syarif Kasim Riau
Jl.Soebrantas KM 15 No.155 Pekanbaru

ABSTRAK

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 saat yang sama. Di dalam proses ini ada kemungkinan terjadi *deadlock. Deadlock* yaitu suatu kondisi ketika dua proses atau lebih tidak dapat meneruskan eksekusinya. Penulis bermaksud memberikan solusi untuk masalah tersebut dengan merancang suatu perangkat lunak yang dapat mensimulasikan proses terjadinya *deadlock* dan sekaligus mensimulasikan pencegahan masalah *deadlock*. Salah satu metode yang dapat menggambarkan permasalahan tersebut adalah *Dining philosophers problems*. *Dining philosophers problems* diilustrasikan sebagai berikut terdapat 5 (lima) orang filsuf yang duduk mengelilingi sebuah meja. Masing-masing filsuf mempunyai mangkuk yang berisi mie dan sebuah sumpit untuk tiap filsuf. Para filsuf menghabiskan waktu dengan berpikir ketika kenyang dan makan ketika lapar. Jika filsuf akan makan mie filsuf harus mempunyai dua buah sumpit ditangannya. Simulasi ini diharapkan dapat membantu pemahaman tentang *deadlock* dan cara pencegahannya.

Kata kunci : *Deadlock*, *Dining Philosophers Problems*, Simulasi.

DAFTAR ISI

Halaman

DAFTAR ISTILAH

Form	Bentuk dari sebuah tampilan
Implementasi	Pelaksanaan atau penerapan
Input	Data yang dimasukkan
Interface	Tampilan antar muka
Komponen	Bagian dari keseluruhan atau unsur
Output	Data yang dihasilkan
User	Pemakai
Hardware	Sebuah alat/benda yang kita bisa lihat, sentuh, pegang dan memiliki fungsi tertentu
Software	Sekumpulan data elektronik yang disimpan dan diatur oleh komputer
Proses	Unit kerja terkecil yang secara individu memiliki sumber daya-sumber daya dan dijadwalkan sistem operasi
Efisien	*P*enggunaan sumber daya secara minimum guna pencapaian hasil yang optimum.
CPU	Suatu perangkat keras microprocessor yang memahami dan melaksanakan suatu perintah dari perangkat lunak
I/O *device*	*B*agian dari sistem mikroprosesor yang digunakan oleh mikroprosesor itu untuk berhubungan dengan dunia luar
Resource	Sumber daya-sumber daya dan dijadwalkan sistem operasi
***Process Control Block* (PCB)**	Manifestasi dari suatu proses dalam suatu sistem operasi
Multithreading	Sistem yang memungkinkan lebih dari satu *thread* dieksekusi secara bersamaan

Sinkronisasi	Proses pengaturan jalannya beberapa proses pada saat yang bersamaan
Mutual Exclusion	Adalah jaminan hanya satu proses yang mengakses sumber daya pada suatu interval waktu tertentu.
Simulasi	Proses merancang model dari suatu sistem yang sebenarnya,

BAB I
PENDAHULUAN

1.1 Latar Belakang

Suatu proses disebut *deadlock* apabila terjadi dua proses atau lebih tidak dapat meneruskan eksekusinya. Permasalahan *deadlock* terjadi karena sekumpulan proses-proses yang diblok di mana setiap proses membawa sebuah sumber daya dan menunggu mendapatkan sumber daya yang dibawa oleh proses lain.

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 saat yang sama. *Deadlock* merupakan masalah yang sering kita jumpai pada setiap sistem operasi, akan tetapi permasalahan ini sudah ada solusi dan pencegahannya.

Ada beberapa metode untuk mencegah terjadinya *deadlock* salah satunya adalah dengan menggunakan metode *Dining Philosophers Problem* dapat diilustrasikan sebagai berikut, terdapat lima orang filsuf yang akan makan mie ayam. Di meja disediakan lima buah sumpit. Jika filsuf benar-benar lapar, 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.

Untuk itu penulis bermaksud untuk merancang suatu perangkat lunak yang dapat mensimulasikan proses kerja dari problema tersebut sekaligus mensimulasikan pencegahan masalah *deadlock*.

1.2 Rumusan Masalah

Berdasarkan latar belakang di atas, maka yang menjadi permasalahan adalah bagaimana kita membuat suatu simulasi yang menggambarkan proses

terjadinya *deadlock* dan pencegahannya dengan menggunakan *dinning philosophers problems.*

1.3 Batasan Masalah

Ruang lingkup atau yang menjadi permasalahan dalam merancang perangkat lunak ini adalah sebagai berikut:

1. Pembahasan permasalahan yang terjadi pada manajemen proses.
2. Penyelesaiannya menggunakan metode *dinning philosopher problems.*

1.4 Tujuan Penelitian

Adapun tujuan dari penelitian penyusunan tugas akhir ini adalah sebagai berikut :

1. Agar mahasiswa mengerti dan dapat memahami serta mendalami sistem operasi secara teoritis dan praktis.
2. Agar mahasiswa mengerti dan memahami proses terjadinya keadaan *deadlock* dan pencegahannya dengan simulasi *Dining Philosophers problems*.
3. Sebagai fasilitas tambahan dalam proses belajar mengajar terutama dalam mata kuliah Sistem Operasi.

1.5 Sistematika Pembahasan

Sistematika pembahasan tugas akhir ini dibagi menjadi 6 (enam) bab. Setiap bab terdiri dari subbab dan penjelasan yang tersusun sehingga mudah untuk dipahami. Berikut penjelasan tentang masing-masing bab:

BAB I Pendahuluan

Merupakan deskripsi umum dari tugas akhir ini, yang meliputi: latar belakang masalah, rumusan masalah, batasan masalah, tujuan penyusunan tugas akhir serta sistematika pembahasan tugas akhir.

BAB II Landasan Teori

Berisi penjelasan tentang teori dasar yang akan diterapkan dalam tahap analisis sistem menggunakan simulasi *Dining Philosopher Problems,*

penjelasan sistem operasi dan cara kerjanya serta sebab-sebab terjadinya *deadlock* pada sistem operasi.

BAB III Metodologi Penelitian

Dalam bab ini menjelaskan mengenai pengamatan pendahuluan, analisa, perancangan, implementasi, pengujian dan kesimpulan dan saran.

BAB IV Analisis dan Perancangan

Bab ini membahas menganalisa tentang prinsip kerja *dining philosophers problems* dan perancangan simulasi yang akan dibuat.

BAB V Implementasi dan Pengujian

Pada bab ini akan dibahas lingkungan implementasi, hasil implementasi, dan pengujian dari simulasi.

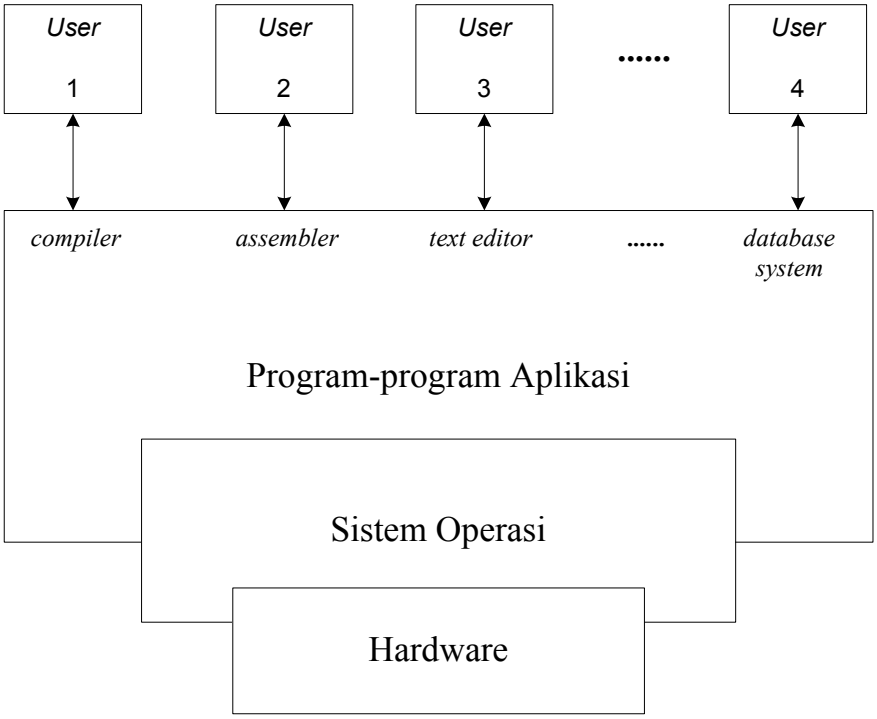
BAB VI Penutup

Dalam bab ini akan dijelaskan beberapa kesimpulan yang didapatkan dari pembahasan tentang simulasi serta saran untuk pengembangan selanjutnya.

BAB II
LANDASAN TEORI

2.1 Sistem Operasi

Sistem operasi adalah bagian yang sangat penting bagi semua sistem komputer (Sri Kusuma Dewi, 2000). Secara umum, sistem komputer terbagi atas *hardware*, sistem operasi, program aplikasi, dan *user*. *Hardware* terdiri atas CPU, memori dan I/O *device* yang merupakan *resource-resource* dasar. Program aplikasi berisi basis data, *games* dan program-program bisnis, yang merupakan suatu cara atau alat dimana *resource-resource* akan diakses untuk menyelesaikan masalah *user*. Komponen-komponen sistem komputer tersebut ditunjukkan oleh gambar 2.1 berikut ini :



Gambar 2.1 Komponen – komponen sistem komputer

2.1.1 Definisi Sistem Operasi

Ada beberapa definisi yang dapat diberikan untuk sistem operasi, antara lain:

a. *Software* yang mengontrol *hardware*, hanya berupa program biasa (seperti beberapa *file* pada DOS).
b. Program yang menjadikan *hardware* lebih mudah untuk digunakan.
c. Kumpulan program yang mengatur kerja *hardware* (seperti: mengatur memori, *printer* dll).
d. *Resource Manager* atau *resource allocator* (seperti: mengatur memori, printer dll).
e. Sebagai program pengontrol (program yang digunakan untuk mengontrol program yang lainnya).
f. Sebagai Kernel, yaitu program yang terus-menerus *running* selama komputer dihidupkan.
g. Sebagai *guardian*, yaitu mengatur atau menjaga komputer dari berbagai kejahatan komputer.

2.1.2 Peranan Sistem Operasi

Peranan sistem operasi pada komputer adalah sebagai berikut:

a. Sebagai antarmuka (*interface* atau jembatan penghubung) antara *user* dengan *hardware*.
b. Memungkinkan adanya pemakaian bersama *hardware* maupun data antar *user*.
c. Pengatur penjadwalan *resource* bagi *user* (seperti: pemakaian CPU dan I/O secara bergantian, dengan adanya memori *manager* dapat mengakses program besar hanya dengan memori kecil).
d. Menyediakan fasilitas sistem operasi (seperti: menyediakan fasilitas *interrupt*).

2.1.3 Tujuan Adanya Sistem Operasi

Tujuan adanya sistem operasi, yaitu:

a. Menunjukkan lingkungan dimana seorang *user* dapat mengeksekusi program-programnya.
b. Membuat sistem komputer nyaman untuk digunakan.
c. Mengefisienkan *hardware* komputer.

2.1.4 Sejarah Singkat Perkembangan Sistem Operasi

(Bambang Hariyanto, 2005) Perkembangan sistem operasi sangat dipengaruhi oleh perkembangan *hardware*, yaitu:

1. Generasi ke-nol (1940)
    a. Komponen utama berupa tabung hampa udara.
    b. Sistem komputer belum menggunakan sistem operasi.
    c. Semua operasi komputer dilakukan secara manual melalui *plugboards*, dan hanya bisa digunakan untuk menghitung (+, - dan *).
2. Generasi ke-pertama (1950)
    a. Komponen utama berupa transistor.
    b. Sistem operasi berfungsi terutama sebagai pengatur pergantian antar *job* agar waktu instalasi *job* berikutnya lebih efisien. Dalam masa ini muncul konsep *batch system* (semua *job* sejenis dikumpulkan jadi satu).
    c. Input memakai *punch card.*
3. Generasi ke-dua (1960)
    a. Komponen utama berupa IC.
    b. Berkembang konsep-konsep:
        a. *Multiprogramming*, satu prosesor mengerjakan banyak program yang ada di memori utama.
        b. *Multiprocessing,* satu *job* dikerjakan oleh banyak prosesor untuk meningkatkan utilitas.
        c. *Spooling (Simultaenous Peripheral Operation On Line),* bertindak sebagai buffer saja, dan mampu menerima pesanan meskipun belum akan dikerjakan.
        d. *Device Independence,* masing-masing komponen memiliki sifat yang saling berbeda (misal: tiap-tiap *printer* memiliki *driver*).
        e. *Time Sharing* atau *multitasking.*
        f. *Real time system,* berguna sebagai kontrol bagi mesin-mesin.
4. Generasi ke-tiga (1970)

a. Komponen utama berupa VLSI (*Very Larga Scale Integrated Sircuit*).
b. Ditandai dengan berkembangnya konsep *general purpose system*, sehingga sistem operasi menjadi sangat kompleks, mahal dan sulit dipelajari.

5. Generasi ke-empat (pertengahan 1970-an hingga sekarang)
   a. PC makin populer.
   b. Ditandai dengan berkembangnya sistem operasi untuk jaringan komputer dengan tujuan: *data sharing*, *hardware sharing* dan *program sharing*.
   c. *User interface*, semakin *user friendly* tanpa harus mengorbankan kinerja sistem.

2.1.5 Sudut Pandang Sistem Komputer

Sudut pandang sistem komputer dapat dikelompokkan menjadi tiga yaitu :

1) Pemakai terdiri dari pemakai awam (*end user*) dan administrator sistem.

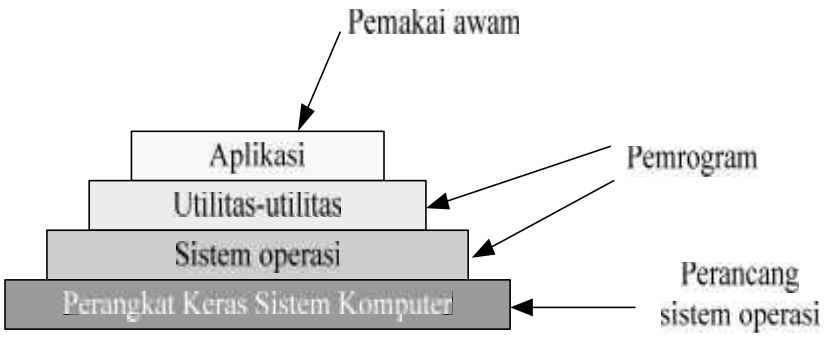
   Pemakai awam menggunakan aplikasi tertentu, tidak berkepentingan dengan arsitektur komputer. Pemakai memandang sistem komputer dalam antarmuka yang disediakan aplikasi. Aplikasi dikembangkan pemrogram aplikasi. Administrator bertugas agar sistem berfungsi secara benar dan optimal.

2) Pemrogram.

   Bila pemrogram aplikasi mengembangkan aplikasi dengan menggunakan kumpulan instruksi mesin secara langsung, maka pemrogram bertanggung jawab sepenuhnya untuk mengendalikan perangkat keras komputer. Tugas ini amat kompleks. Untuk mempermudah pemrogram, sekumpulan fasilitas disediakan sistem operasi. Pemrogram menggunakan fasilitas ini dalam mengembangkan aplikasi, mengelola berkas, dan mengendalikan masukan / keluaran, tak perlu menulis atau membuat sendiri.

Sistem operasi menyembunyikan rincian operasi perangkat keras dan menyediakan antarmuka yang nyaman untuk menggunakan perangkat. Sistem operasi bertindak sebagai mediator, mempermudah pemrogram. Sistem operasi juga memberi cara program aplikasi mengakses dan menggunakan fasilitas-fasilitas dan layanan-layanan sistem komputer.

Pandangan terhadap sistem komputer sebagai berlapis atau hirarki seperti terlihat pada gambar 2.2 di bawah ini :



Gambar 2.2 Hirarki Pandangan terhadap Sistem Komputer.

3) Perancang sistem operasi.

Perancang sistem operasi harus membuat sistem operasi yang dapat mempermudah dan menyamnakan terutama untuk pemrogram aplikasi membuat aplikasi-aplikasi.

2.2 Struktur Sistem Operasi

2.2.1 Komponen-komponen Sistem Operasi

(Kusnadi, 2008) Sistem operasi memiliki komponen-komponen sebagai berikut,

1. Manajemen Proses

   Sistem operasi memberikan tanggapan terhadap manajemen proses untuk aktivitas-aktivitas sebagai berikut:

   a. Pembuatan atau penghapusan proses yang dibuat oleh *user* atau sistem.

b. Suspensi dan asumsi proses.
c. Kelengkapan mekanisme untuk sinkronisasi proses.
d. Kelengkapan mekanisme untuk komunikasi proses.
e. Kelengkapan mekanisme untuk pengendalian *deadlock*.

2. Manajemen Memori Utama

   Sistem operasi memberikan tanggapan terhadap manajemen memori utama untuk aktivitas-aktivitas sebagai berikut:

   a. Menjaga dan memelihara bagian-bagian memori yang sedang digunakan dan dari yang menggunakan.
   b. Memutuskan proses-proses mana saja yang harus dipanggil ke memori jika masih ada ruang di memori.
   c. Mengalokasikan dan mendealokasikan ruang memori jika diperlukan.

3. Manajemen Memori Sekunder

   Sistem operasi memberikan tanggapan terhadap manajemen penyimpanan sekunder untuk aktivitas-aktivitas sebagai berikut:

   a. Pengaturan ruang kosong.
   b. Alokasi penyimpanan.
   c. Penjadwalan *disk*.

4. Manajemen I/O

   Sistem operasi memberikan tanggapan terhadap manajemen I/O untuk aktivitas-aktivitas sebagai berikut:

   a. Sistem *buffer-chaching*.
   b. Antarmuka *device-driver* secara umum.
   c. *Driver* untuk *device hardware-hardware* tertentu.

5. Manajemen File

   Sistem operasi memberikan tanggapan terhadap manajemen *file* untuk aktivitas – aktivitas sebagai berikut:

   a. Pembuatan dan penghapusan *file*.
   b. Pembuatan dan penghapusan direktori.

c. Primitif-primitif yang mendukung untuk manipulasi *file* dan direktori.
d. Pemetaan *file* ke memori sekunder.
e. *Backup file* ke media penyimpanan yang stabil (*nonvolatil*).

2.2.2 Pelayanan Sistem Operasi

Sistem operasi harus dapat melayani *programmer* sehingga dapat melakukan pemograman dengan mudah.

a. Eksekusi Program.

   Sistem harus dapat memanggil program ke memori dan menjalankannya. Program tersebut harus dapat mengakhiri eksekusinya dalam bentuk normal atau abnormal (indikasi *error*).

b. Operasi-Operasi I/O.

   Pada saat *running program* kemungkinan dibutuhkan I/O, mungkin berupa *file* atau peralatan I/O. Agar efisien dan aman, maka *user* tidak boleh mengontrol I/O secara langsung, pengontrolan dilakukan oleh sistem operasi.

c. Manipulasi Sistem *File*.

   Meliputi pembuatan, penghapusan, *read* dan *write*.

d. Komunikasi.

   Komunikasi dibutuhkan jika beberapa proses saling tukar-menukar informasi. Ada 2 cara yang dapat dilakukan, yaitu tukar-menukar data oleh beberapa proses dalam satu komputer dan tukar-menukar data oleh beberapa proses dalam komputer yang berbeda melalui sistem jaringan. Komunikasi dilakukan dengan cara berbagi memori atau dengan cara pengiriman pesan.

e. Mendeteksi Kesalahan.

   Untuk masing-masing kesalahan, sistem operasi harus memberikan aksi yang cocok agar komputasinya menjadi konsisten.

2.3 *Dining Philosophers Problems*

Pada tahun 1965, Dijkstra menyelesaikan sebuah masalah klasik dalam sinkronisasi yang beliau namakan *dining philosophers problems* (Abas Ali Pangera, Dony Ariyus, 2010). *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 untuk sementara waktu. 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.

Prosedur *take-fork* menunggu sampai sumpit-sumpit yang sesuai didapatkan dan kemudian menggunakanya, sayangnya dari solusi ini ternyata salah. Seharusnya lima orang filsuf mengambil sumpit kirinya secara bersamaan. Tidak akan mungkin mereka mengambil sumpit kanan dan akan terjadi *deadlock.*

Pertanyaan kuncinya adalah dapatkah kita membuat program untuk masing-masing filsuf melakukan apa yang harus mereka lakukan dan tidak mengalami kebuntuan.

Kita dapat memodifikasi program sehingga mengambil sumpit kiri, program memeriksa apakah sumpit kanan memungkinkan untuk diambil. Jika sumpit kanan tidak memungkinkan untuk diambil maka filsuf akan meletakkan kembali sumpit kirinya, menunggu untuk beberapa waktu kemudian mengulangi proses yang sama. Usulan tersebut juga salah, walau dengan alasan yang berbeda. Dengan sedikit nasib buruk, semua filsuf dapat memulai algoritma secara bersamaan, mengambil sumpit kiri mereka, melihat sumpit kanan yang tidak mungkin diambil, meletakkan sumpit kiri mereka, menunggu, mengambil sumpit keri mereka secara bersamaan dan begitu seterusnya. Situasi seperti ini berjalan terus tetapi tidak ada perubahan/kemajuan yang dihasilkan disebut *starvation.*

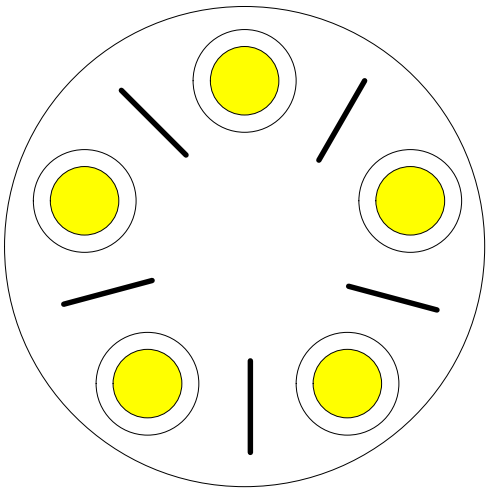
Sekarang kita dapat berfikir “jika filsuf dapat saja menunggu sebuah waktu acak sebagai pengganti waktu yang sama setelah tidak dapat mengambil sumpit

kiri dan kanan, kesempatan bahwa segala sesuatu akan berlanjut dalam kemandegan untuk beberapa jam adalah sangat kecil." Pemikiran seperti itu adalah benar, tapi beberapa aplikasi mengirimkan sebuah solusi yang selalu bekerja dan tidak ada kesalahan, tidak seperti nomor acak yang selalu berubah.

Sebelum mulai mengambil sumpit, filsuf akan melakukan Down di mutex. Setelah menggantikan sumpit dia harus melakukan Up di mutex. Dari segi teori solusi ini cukup memadai. Dari segi praktik solusi ini memiliki masalah. Hanya ada satu filsuf yang dapat makan mie dalam berbagai kesempatan. Dengan lima buah sumpit, seharusnya ada dua orang filsuf yang makan pada saat yang bersamaan.

Solusi yang diberikan diatas benar dan juga mengizinkan jumlah maksimum kegiatan paralel untuk sebuah sejumlah filsuf yang berubah-ubah ini menggunakan sebuah *array*, state, untuk merekam status seorang filsuf apakah sedang makan (*eating*), berfikir (*think*), atau sedang lapar (*hungry*) karena sedang mengambil sumpit. Seorang filsuf hanya dapat berstatus makan jika tidak ada tetangganya yang sedang makan juga. Tetangga seorang filsuf didefinisikan oleh *LEFT* dan *RIGHT*.

Dengan kata lain, jika i=2, maka tetangga kirinya (*LEFT*) = 1 dan tetangga kanannya (*RIGHT*) = 3. Program ini menggunakan sebuah *array* dari *semaphore* yang lapar dapat ditahan jika sumpit kiri atau kanannya sedang dipakai tetangganya. Catatan bahwa masing-masing proses menjalankan prosedur filsuf sebagai kode utama, tetapi prosedur yang lain seperti *take-fork* dan test adalah prosedur biasa dan bukan proses-proses yang terpisah.



Gambar 2.3 Lima filsuf dalam satu meja makan

Meskipun solusi ini menjamin bahwa tidak ada 2 tetangga yang makan bersama-sama, namun masih mungkin terjadi *deadlock*, yaitu 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*. Ada beberapa cara untuk menghindari *deadlock*, antara lain:

a. Mengizinkan paling banyak 4 orang filsuf yang duduk bersama-sama pada satu meja.
b. Mengizinkan seorang filsuf mengambil sumpit hanya jika kedua sumpit itu ada.
c. Menggunakan suatu solusi asimetrik, yaitu filsuf pada nomor ganjil mengambil sumpit kiri dulu baru sumpit kanan. Sedangkan filsuf yang duduk di kursi genap mengambil sumpit kanan dulu baru sumpit kiri.

2.4 *Deadlock*

Proses disebut *deadlock* jika proses menunggu satu kejadian tertentu yang tak akan pernah terjadi (Bambang Hariyanto, 2005). Sekumpulan proses berkondisi *deadlock* bila setiap proses yang ada di kumpulan itu menunggu suatu kejadian yang hanya dapat dilakukan proses lain yang juga berada di kumpulan itu. Proses menunggu kejadian yang tidak akan pernah terjadi. *Deadlock* terjadi ketika proses-proses mengakses secara ekslusif sumber daya. Semua *deadlock*

yang terjadi melibatkan persaingan memperoleh sumber daya ekslusif oleh dua proses atau lebih.

Terjadi *trade-off* antara *overhead* mekanisme penyelesaian *deadlock* dan manfaat-manfaat yang diperoleh. Pada beberapa kasus, ongkos yang harus dibayar untuk membebaskan sistem dari *deadlock* adalah mahal. Seperti pada sistem pengendalian proses waktu nyata (*real-time process control system*), tak ada pilihan lain kecuali membayar semua kemahalan karena adanya *deadlock* akan mengakibatkan sistem kacau.

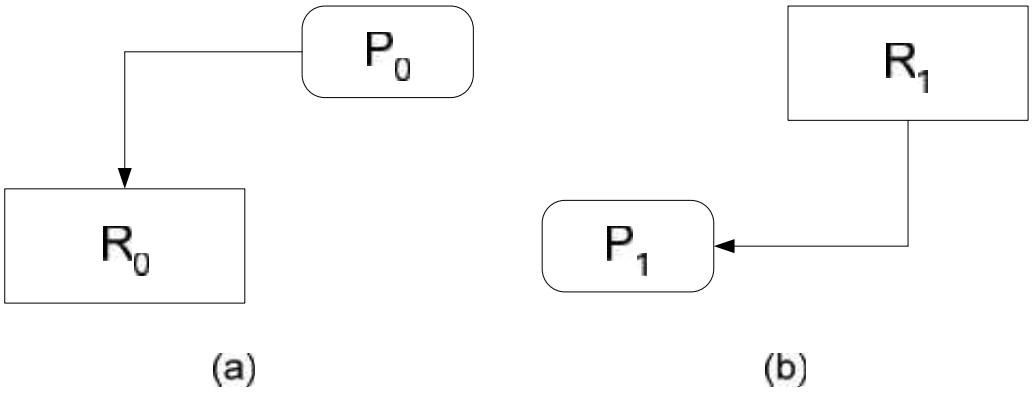
Proses dikatakan sebagai mengalami *starvation* bila proses-proses itu menunggu alokasi sumber daya sampai tak berhingga, sementara proses-proses lain dapat memperoleh alokasi sumber daya. *Starvation* disebabkan bias pada kebijaksanaan atau strategi alokasi sumber daya. Kondisi ini harus dihindari karena tidak adil tetapi dikehendaki penghindaran dilakukan seefisien mungkin.

2.4.1 Model *Deadlock*

Urutan kejadian pengoperasian perangkat masukan/keluaran adalah:

- Meminta (*request*), yaitu meminta pelayanan perangkat masukan / keluaran.
- Memakai (*use*), yaitu memakai perangkat masukan/keluaran.
- Melepaskan (*release*), yaitu melepaskan pemakaian perangkat masukan / keluaran.

Misalnya, terdapat dua proses P_0 dan P_1 dan dua sumber daya R_0 dan R_1.

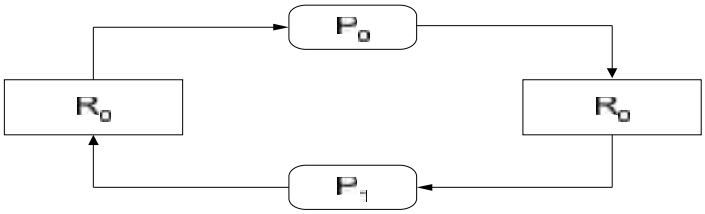


Gambar 2.4 *Graph* meminta sumber daya dan alokasi sumber daya.

Gambar 2.4 (a) P_0 meminta sumber daya R_0, ditandai busur (*edge*) berarah dari proses P_0 ke sumber daya R_0. Gambar 2.4 (b) sumber daya R_1 dialokasikan ke P_1, ditandai busur berarah dari sumber daya R_1 ke proses P_1. Kemudian terjadi skenario berikut,

- P_0 sambil masih menggenggam R_0, meminta R_1.
- P_1 sambil masih menggenggam R_1, meminta R_0.

Kejadian ini mengakibatkan *deadlock* karena sama-sama akan saling menunggu. *Deadlock* tidak hanya terjadi pada dua proses dan dua sumber daya, *deadlock* dapat terjadi dengan melibatkan lebih dari dua proses dan dua sumber daya.



Gambar 2.5 *Graph deadlock* dua proses dan dua sumber daya

2.4.2 Metode – Metode Mengatasi *Deadlock*

Terdapat empat syarat untuk terjadinya *deadlock*, yaitu:

1. *Mutual Exclution Condition*

   Tiap sumber daya saat itu diberikan pada tepat satu proses.
2. Kondisi genggam dan tunggu (*hold and wait condition*)

   Proses-proses yang sedang mengenggam sumber daya, menunggu sumber daya-sumber daya baru.
3. Kondisi *non-preemption* (*non-preemption condition*)

   Sumber daya-sumber daya yang sebelumnya diberikan tidak dapat diambil paksa dari proses itu. Sumber daya-sumber daya harus secara eksplisit dilepaskan dari proses yang mengenggamnya.
4. Kondisi menunggu secara sirkuler (*circular wait condition*)

Harus terdapat rantai sirkuler dari dua proses atau lebih, masing-masing menunggu sumber daya yang digenggam oleh anggota berikutnya pada rantai itu.

Ketiga syarat pertama merupakan syarat perlu (*necessary conditions*) bagi terjadinya *deadlock*. Keberadaan *deadlock* selalu berarti terpenuhi kondisi-kondisi di atas, tak mungkin terjadi *deadlock* bila tidak ada ketiga kondisi itu. *Deadlock* terjadi berarti terdapat ketiga kondisi itu, tetapi adanya ketiga kondisi itu belum berarti terjadi *deadlock*. *Deadlock* baru benar-benar terjadi bila syarat keempat terpenuhi. Kondisi keempat merupakan keharusan bagi terjadinya peristiwa *deadlock*. Bila salah satu kondisi saja tidak terpenuhi, maka *deadlock* tidak terjadi.

2.4.3 Pencegahan *Deadlock*

Havender, J.W dalam bukunya berjudul '*Avoiding Deadlock in Multitasking Systems*' mengemukakan jika sembarang syarat dari keempat syarat tak terpenuhi maka tidak akan terjadi *deadlock*. Havender menyarankan starategi-strategi berikut untuk meniadakan syarat-syarat tersebut, yaitu:

1. Tiap proses harus meminta sumber daya yang diperlukan sekaligus dan tidak berlanjut sampai semuanya diberikan.
2. Jika proses telah sedang memegang sumber daya tertentu, untuk permintaan berikutnya proses harus melepas dulu sumber daya yang dipegangnya. Jika diperlukan, proses meminta kembali sekaligus dengan sumber daya yang baru.
3. Beri pengurutan linear terhadap tipe-tipe sumber daya pada semua proses, yaitu jika proses telah dialokasikan suatu tipe sumber daya, proses hanya boleh meminta sumber daya-sumber daya tipe pada urutan yang berikutnya.

2.5 Model

Model adalah representasi dari suatu objek (benda) atau ide-ide dalam bentuk yang lain. Model juga dapat didefenisikan sebagai suatu gambaran,

abstraksi atau imajinasi suatu sistem nyata. Ataupun berupa suatu abstraksi dunia nyata yang pada akhirnya dapat digunakan untuk mengambil keputusan.

Model berisi informasi-informasi tentang sesuatu yang dibuat dengan tujuan untuk mempelajari sistem yang sebenarnya. Model dapat berupa tiruan dari suatu benda, sistem atau peristiwa sesungguhnya yang hanya mengandung informasi-informasi yang dipandang penting untuk ditelaah.

Model yang dibuat dapat berguna untuk :

1. Membantu dalam berpikir, model menyajikan deskripsi yang sistematis tentang suatu sistem sehingga dapat mempermudah mempelajari sistem tersebut.
2. Membantu untuk berkomunikasi atau mempermudah menjelaskan tentang suatu sistem kepada orang lain.
3. Sebagai alat latihan, untuk melatih ketrampilan orang-orang yang berhubungan dengan sistem sebenarnya yang dimodelkan. Contohnya : Simulator dalam dunia penerbangan, ini digunakan untuk melatih seorang calon pilot yang dalam taraf belajar, belum boleh mengemudikan pesawat yang sebenarnya, tetapi belajar mengemudikan suatu model yang mewakili pesawat dan juga mengoperasikan model tersebut terhadap suatu model lapangan terbang, udara, lingkungan terbang dan sebagainya.
4. Sebagai alat prediksi terhadap kelakuan sistem untuk waktu yang akan datang, yaitu pengaruh-pengaruh yang ingin diketahui jika ada perubahan sistem atau operasi sistem.
5. Membantu dalam melakukan percobaan, dalam hal melakukan percobaan atau eksperimen tidak mungkin langsung dilaksanakan atau diadakan secara praktis karena biaya yang mahal dan bahaya atau resiko yang tinggi.

Dalam pembuatan model, proses pembuatan model tidak dapat digambarkan secara pasti, namun ada petunjuk yang dapat digunakan yaitu sebagai berikut :

1. Pemecahan masalah melalui penyederhanaan.
2. Menyatakan objek dengan persyaratan yang jelas karena objek sangat menentukan model.

3. Mencari analog-analog dan sistem atau model yang sudah ada untuk mempermudah konstruksi.
4. Menentukan komponen-komponen yang akan dimasukkan ke dalam model.
5. Menentukan variabel, konstanta dan parameter, hubungan fimgsional serta konstrain dari fungsi-fungsi kriterianya.
6. Untuk membuat model matematik harus dipikirkan cara untuk menyatakan masalah secara numerik jika ingin disimulasikan dengan komputer.
7. Nyatakan dengan simbol-simbol.
8. Menuliskan persamaan matematikanya.
9. Bila model terlalu rumit, terdapat beberapa cara untuk menyederhanakan model seperti :
    a. Buat harga variabel menjadi parameter.
    b. Eliminasi / kombinasikan variabel-variabel.
    c. Asumsikan linieritas.
    d. Tambahkan asumsi dan batasan yang ketat.
    e. Perjelas batasan sistem.

Sebelum menentukan model yang akan dibuat, lebih dahulu perlu mempelajari sistemnya. Sistem yang ada seringkali sangat kompleks, tapi model diusahakan dibuat sesederhana mungkin. Salah satu cara untuk mempelajari sistem adalah dengan menuangkan informasi-informasi dari sistem tersebut ke dalam bentuk diagram.

Untuk menilai model apakah dapat dianggap baik sebenarnya cukup sulit, tetapi pada dasarnya kriteria suatu model yang baik dapat diuraikan sebagai berikut :

1. Mudah dimengerti pemakaiannya
2. Harus mempunyai tujuan yang jelas
3. Dinyatakan secara jelas dan lengkap
4. Mudah dikontrol dan dimanipulasi oleh pemakai
5. Mengandung pemecahan masalah yang penting dan jelas
6. Mudah diubah dan mempunyai prosedur modifikasi

7. Dapat berkembang dari sederhana menjadi kompleks.

2.6 Simulasi

Simulasi adalah proses merancang model dari suatu sistem yang sebenarnya, mengadakan percobaan-percobaan terhadap model tersebut dan mengevaluasi hasil percobaan tersebut. Adapun manfaat dari simulasi adalah sebagai berikut :

1. Menjelaskan kelakuan sistem.
2. Menirukan bekerjanya suatu sistem melalui melalui suatu model.
3. Memecahkan suatu persoalan matematik dengan analisis numerik.
4. Mempelajari dinamika suatu sistem.
5. Memberikan suatu deskripsi perilaku sistem dalam perkembangan sejalan dengan bertambahnya waktu.
6. Membangun teori atau hipotesa yang mempertanggungjawabkan kelakuan dari sistem yang diamati.
7. Meramalkan kelakuan sistem yang akan datang yaitu pengaruh yang dihasilkan oleh perubahan-perubahan sistem atau perubahan operasinya.

BAB III
METODOLOGI PENELITIAN

Dalam penelitian tugas akhir ini, metodologi penilitian merupakan pedoman dalam pelaksanaan penelitian sehingga yang dicapai tidak menyimpang dari tujuan yang telah ditentukan sebelumnya.

Dalam metodologi penelitian di jabarkan tahapan-tahapan yang dilakukan dalam penelitian. Metodologi penelitian terdiri dari beberapa tahapan yang terkait secara sistematis. Tahapan ini diperlukan untuk memudahkan dalam melakukan penelitian.

3.1 Pengamatan Pendahuluan

Pengamatan pendahuluan merupakan tahapan awal dalam melakukan penelitian. Tahap ini dilakukan untuk menemukan permasalahan dari masalah sinkronisasi pada *sistem operasi* dan meneliti lebih rinci sehingga akan mempermudah mengelompokan data ditahap berikutnya. Data yang diharapkan diperoleh dari pengamatan ini adalah :

1. Data dari kasus *dining philosophers problems* sebagai salah satu masalah sinkronisasi yang diangkat pada tugas akhir ini.
2. Data mengenai penyebab terjadinya *deadlock* pada kasus pengeksekusian sebuah program menggunakan simulasi *dining philosopher program.*
3. Data mengenai metode yang digunakan untuk menyelesaikan masalah yang dibahas, disini metode yang digunakan adalah metode semaphore.

3.2 Analisa

Pada tahap ini dilakukan analisa permasalahan yang telah dirumuskan, yaitu menganalisa kasus *dining philosopher problem* yang digunakan sebagai ilustrasi pembahasan sinkronisasi dan pengeksekusian sebuah program agar tidak terjadi *deadlock* .

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.

Untuk menghindari kondisi *deadlock*, ada 3 solusi yang dapat dipilih, yaitu:

1. Solusi-A, yaitu mengizinkan paling banyak 4 orang filsuf yang duduk bersama-sama pada satu meja.
2. Solusi-B, yaitu mengizinkan seorang filsuf mengambil sumpit hanya jika kedua sumpit itu ada.
3. Solusi-C, yaitu filsuf pada nomor ganjil mengambil sumpit kiri dulu baru sumpit kanan, sedangkan filsuf pada nomor genap mengambil sumpit kanan dulu baru sumpit kiri. Solusi ini disebut solusi asimetrik.

3.3 Perancangan

Setelah melakukan analisa, maka kemudian dilanjutkan dengan perancangan sistem berdasarkan analisa permasalahan yang telah dilakukan sebelumnya. Perancangan perangkat lunak ”Aplikasi pencegahan deadlock menggunakan simulasi *Dining Philosopher Problem”* menggunakan bahasa pemograman *Microsoft Visual Basic* 6.0. Pembuatan objek simulasi dan animasi menggunakan aplikasi *Adobe Photoshop CS3*.

3.3.1 Perancangan Struktur Menu

Rancangan struktur menu diperlukan untuk memberikan gambaran terhadap menu-menu atau fitur pada sistem yang akan dibangun.

Beberapa komponen standar yang dipakai dalam membangun struktur menu pada aplikasi ini antara lain:

1. *Command button,* untuk menjalankan perintah program
2. *Textbox*, untuk memasukkan *input*.
3. *Checkbox* , untuk memilih opsi keadaan mati untuk filsuf.
4. *image,* untuk mengelompokkan gambar sumpit filsuf.
5. *Timer*, komponen untuk menjalankan proses animasi (memeriksa keadaan dan menggerakkan keadaan ke keadaan berikutnya).

3.3.2 Perancangan Antar Muka (*interface*)

Untuk mempermudah komunikasi antara sistem dengan pengguna, maka perlu dirancang antar muka *(interface)*. Dalam perancangan *interface* hal terpenting yang ditekankan adalah bagaimana menciptakan tampilan yang baik dan mudah dimengerti oleh pengguna. Form pada aplikasi simulasi ini di antaranya :

1. *Form Splash Screen*, berfungsi sebagai *form* pertama yang tampil sewaktu perangkat lunak dijalankan
2. *Form Input*, berfungsi sebagai *form* untuk memasukkan *input* perangkat lunak
3. *Form* Proses Simulasi, berfungsi sebagai *form* untuk mensimulasikan proses *dining philosophers problem*

3.4 Implementasi

Setelah analisa dan perancangan sistem selesai, maka tahap selanjutnya adalah implementasi. Implementasi adalah tahapan dimana dilakukan *coding* atau pengkodean. Untuk implementasi perangkat lunak Simulasi *dining philosopher problem* akan dijalankan dengan menggunakan perangkat keras (*hardware*) yang direkomendasikan sebagai berikut :

1. Prosesor Intel Pentium IV 2.26 GHz.
2. Memory 256 MB.
3. Harddisk 40 GB.
4. VGA *card* 32 MB.
5. Monitor dengan resolusi 1024 X 768 *pixel*.
6. *Keyboard* dan *Mouse*.

Adapun sistem operasi yang direkomendasikan untuk menjalankan aplikasi ini adalah lingkungan sistem operasi MS-Windows NT/2000/XP.

3.5 Pengujian

Pengujian merupakan tahapan dimana aplikasi akan dijalankan, tahap ini diperlukan untuk mengetahui apakah sistem sesuai dengan tujuan yang ingin dicapai.

Pengujian sistem dilakukan dengan cara menggunakan *Black Box*. Pada *Black Box* pengujian ini berfokus pada perangkat lunak untuk mendapatkan serangkaian kondisi input yang seluruhnya menggunakan persyaratan fungsional dalam suatu program.

3.6 Kesimpulan dan Saran

Kesimpulan dan saran merupakan tahap akhir dari penelitian tugas akhir yang dilakukan. Dibagian ini akan diambil kesimpulan berdasarkan hasil dari penelitian yang telah dilakukan serta memberikan saran-saran yang membangun untuk menyempurnakan dan mengembangkan penelitian tugas akhir itu.

BAB IV
ANALISA DAN PERANCANGAN

4.1 Analisa

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.

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 dirancang penulis 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.
5. Untuk menghindari kondisi *deadlock*, penulis menyediakan 3 cara yang dapat dipilih, yaitu:

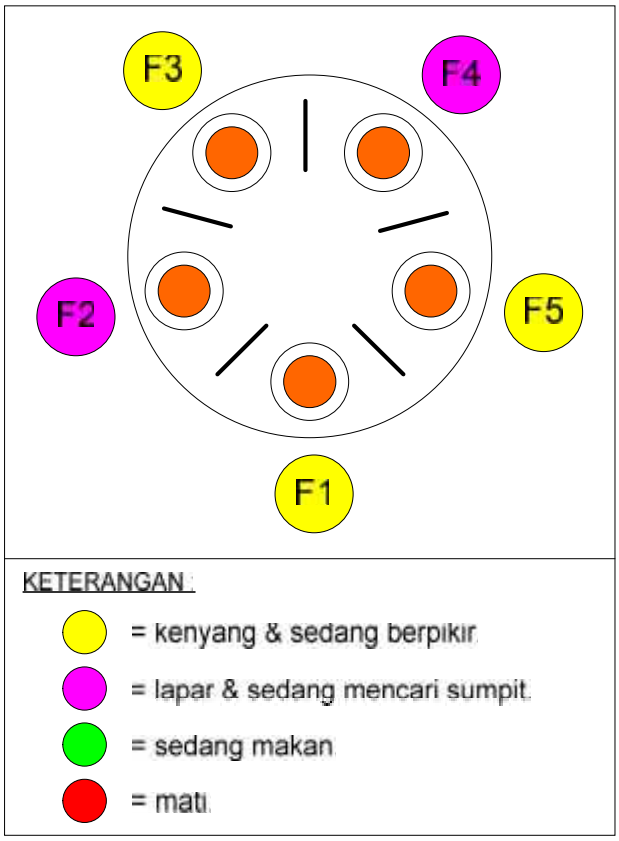
a. Solusi-A, yaitu mengizinkan paling banyak 4 orang filsuf yang duduk bersama-sama pada satu meja.
b. Solusi-B, yaitu mengizinkan seorang filsuf mengambil sumpit hanya jika kedua sumpit itu ada.
c. Solusi-C, yaitu filsuf pada nomor ganjil mengambil sumpit kiri dulu baru sumpit kanan, sedangkan filsuf pada nomor genap mengambil sumpit kanan dulu baru sumpit kiri. Solusi ini disebut solusi asimetrik.

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

Tabel 4.1 Tabel properti filsuf

	Waktu-A (detik)	Waktu-B (detik)	Kondisi Awal (detik)
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

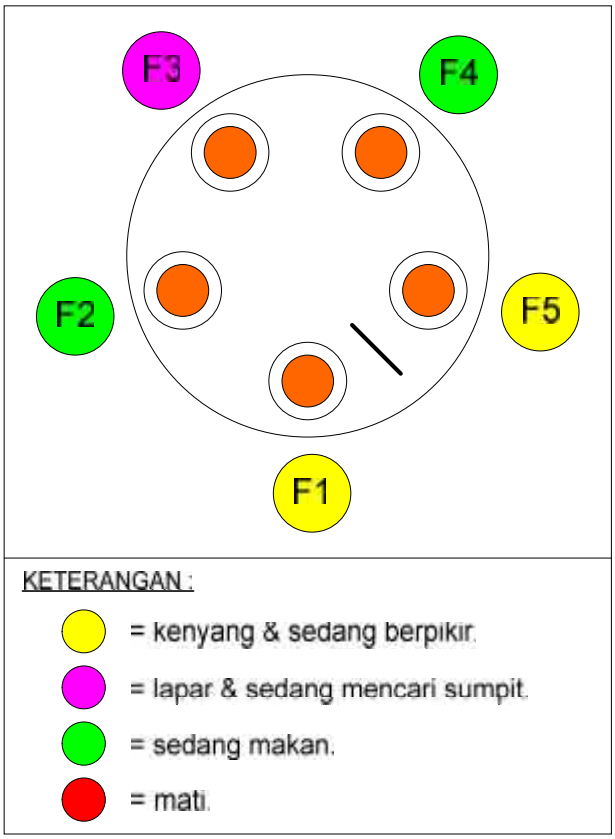
Dari tabel dapat diketahui kondisi masing-masing filsuf, yaitu filsuf-1 berada dalam kondisi kenyang karena kondisi awal 15 detik berada di atas waktu-B yang hanya 10 detik (filsuf-1 akan merasa lapar dan mencari sumpit 5 detik kemudian). Filsuf-2 berada dalam kondisi lapar karena kondisi awal 4 detik = 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 4.1 Bagan illustrasi kondisi awal simulasi

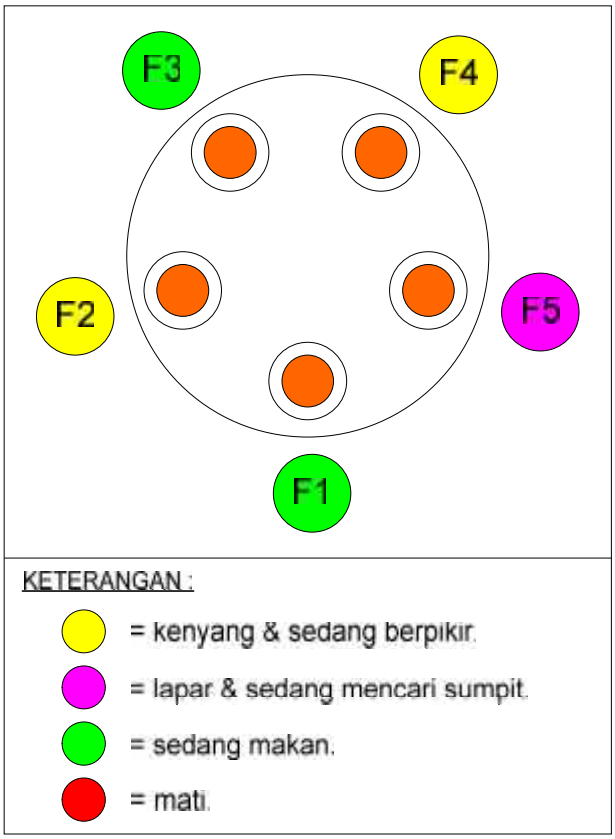
Proses simulasi adalah sebagai berikut:

Pada saat t = 1 detik, 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 detik, 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 detik, filsuf-3 merasa lapar (karena masa hidup filsuf-3 saat ini = waktu-B yaitu 11 detik) 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 4.2 Bagan illustrasi kondisi simulasi saat t=3 detik

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



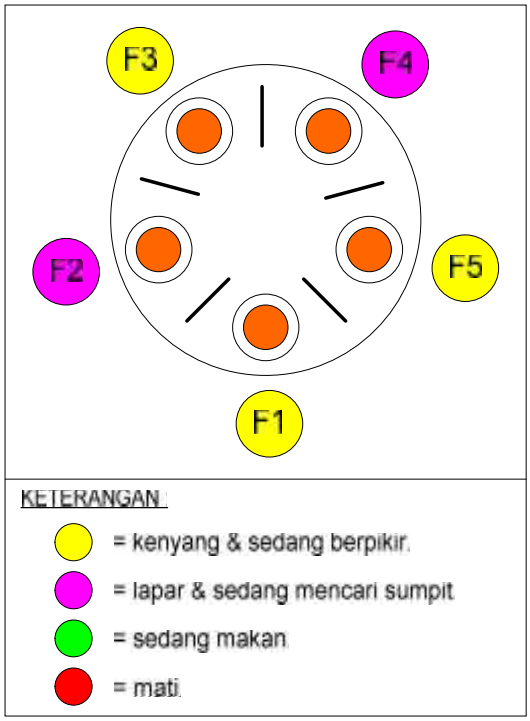
Gambar 4.3 Bagan illustrasi kondisi simulasi saat t=12 detik

Proses simulasi akan berlanjut sesuai dengan prosedur. 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 4.2 Tabel Properti Filsuf untuk Kasus *Deadlock*

	Waktu-A (detik)	Waktu-B (detik)	Kondisi Awal (detik)
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

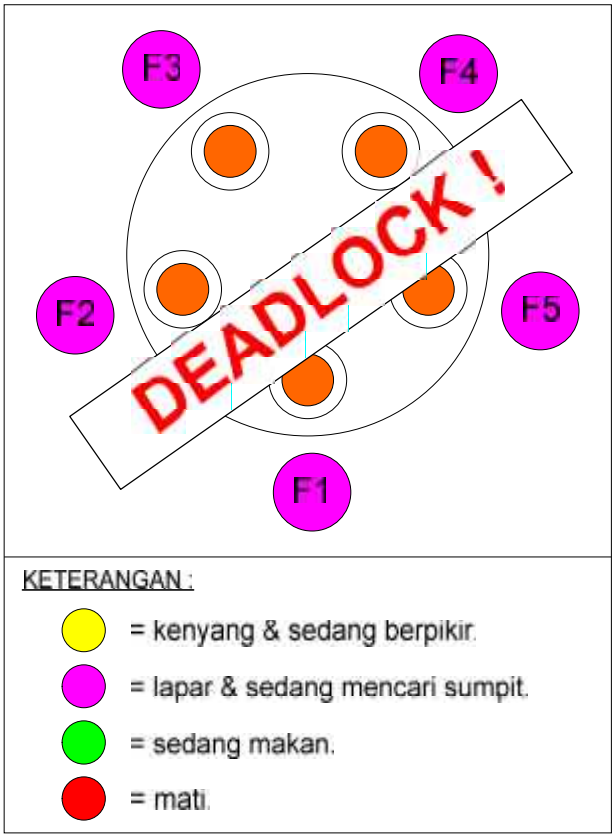
Bagan illustrasi kondisi awal adalah sebagai berikut:



Gambar 4.4 Bagan illustrasi kondisi awal (kasus *deadlock*)

Pada saat t = 1 detik, 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 detik, filsuf-2 dan filsuf-4 mendapat 2 sumpit dan mulai makan. Pada saat t = 10 detik, filsuf-2 dan filsuf-4 kenyang dan mulai berpikir. Pada saat t = 15 detik, 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. Bagan illustrasi kondisi *deadlock* adalah sebagai berikut:



Gambar 4.5 Bagan illustrasi kondisi *deadlock*

Dengan *input* properti filsuf dan kondisi awal yang sama, kondisi *deadlock* yang terjadi dapat dihindari dengan memilih satu dari tiga solusi berikut:

1. Solusi-A, yaitu mengizinkan paling banyak 4 orang filsuf yang duduk bersama-sama pada satu meja. Tidak akan terjadi kondisi *deadlock* apabila terdapat kurang dari 4 orang filsuf yang duduk bersama-sama mengelilingi meja dengan 5 tempat duduk (satu filsuf dianggap mati).
2. Solusi-B, yaitu mengizinkan seorang filsuf mengambil sumpit hanya jika kedua sumpit itu ada. Apabila semua filsuf lapar secara bersamaan, maka

hanya 2 orang filsuf yang dapat makan, karena filsuf mengambil 2 sumpit pada saat yang bersamaan.

3. Solusi-C, yaitu solusi asimetrik. Filsuf pada nomor ganjil mengambil sumpit kiri dulu baru sumpit kanan, sedangkan filsuf pada nomor genap mengambil sumpit kanan dulu baru sumpit kiri. Apabila semua filsuf lapar secara bersamaan, cara pengambilan dengan solusi asimetrik akan mencegah semua filsuf mengambil sumpit kiri secara bersamaan, sehingga kondisi *deadlock* dapat dihindari.

4.2 Perancangan

Perangkat lunak simulasi *dining philosophers* dirancang dengan menggunakan bahasa pemrograman *Microsoft Visual Basic 6.0* dengan beberapa komponen standar seperti *command button, textbox, checkbox, combobox, msflexgrid, shape*, *timer*, *image*, *picturebox* dan komponen lainnya.

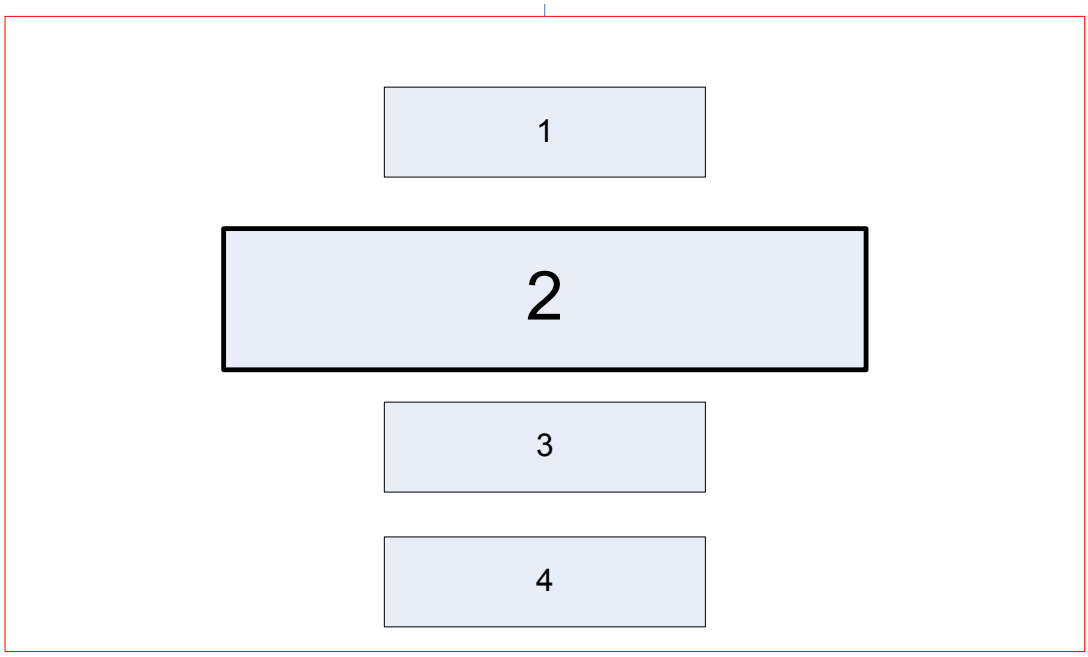
Desain dan gambar yang digunakan di dalam perangkat simulasi ini dirancang dengan menggunakan aplikasi *Adobe Photoshop CS*. Gambar yang telah siap dirancang, ditampilkan di dalam perangkat lunak dengan menggunakan komponen *picturebox*. Untuk animasi perubahan gambar dan proses simulasi, prosedurnya dirancang di dalam komponen *timer*.

Perangkat lunak simulasi ini memiliki beberapa *form*, antara lain:

1. *Form Splash Screen*.
2. *Form Input*.
3. *Form* Proses Simulasi.

4.2.1 Form Splash Screen

Form Splash Screen berfungsi sebagai *form* pertama yang tampil sewaktu perangkat lunak dijalankan. *Form* ini dimaksudkan untuk memberikan informasi awal kepada *user* mengenai perangkat lunak yang sedang dijalankan. Informasi yang ditampilkan mencakup nama perangkat lunak, informasi pembuat perangkat lunak dan nama kampus dimana pembuat menyusun skripsi. *Form* ini akan muncul selama 5 detik sebelum melanjutkan ke *form input*. Namun, apabila *user* ingin menutup *form* ini dan melanjutkan ke proses berikutnya segera, maka *user* dapat menekan sembarang tombol pada *keyboard* atau mengklik *form* di sembarang tempat dengan menggunakan *mouse*.



Gambar 4.6 Rancangan *Form Splash Screen*

Keterangan

1 : informasi masa studi dan jurusan / program studi.

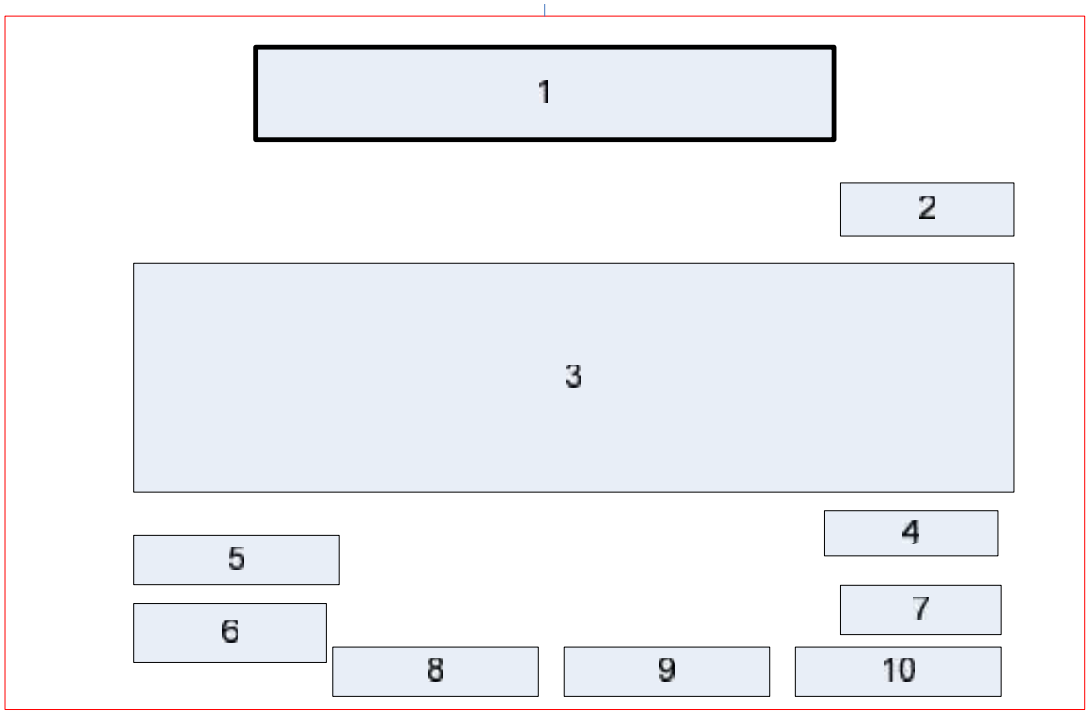
2 : nama perangkat lunak.

3 : nama dan nomor induk mahasiswa

4 : informasi universitas

4.2.2 Form Input

Form Input berfungsi sebagai *form* untuk memasukkan *input* perangkat lunak. *Input* perangkat lunak berupa properti filsuf, tipe pencegahan *deadlock*, opsi '*dead condition available*' dan kecepatan simulasi. Eksekusi *form* ini akan dilanjutkan ke *form* proses simulasi.



Gambar 4.7 Rancangan *Form Input*

Keterangan :

1 : Nama simulasi

2 : *checkbox* 'Dead Condition Available', untuk memilih opsi keadaan mati untuk filsuf.

3 : tabel properti filsuf.

4 : tombol 'Ambil Nilai Acak', untuk mengisi properti filsuf dengan nilai acak dari komputer. Hanya dua(2) digit.

5 : *combobox* 'Pencegahan Deadlock', untuk memilih tipe pencegahan *deadlock* pada proses simulasi. Tipe pencegahan *deadlock* yang dapat dipilih adalah:

a) ‘SOLUSI-A’, yaitu mengizinkan paling banyak 4 filsuf yang duduk bersama-sama pada satu meja.
b) ‘SOLUSI-B’, yaitu mengizinkan seorang filsuf mengambil sumpit hanya jika kedua sumpit itu ada.
c) ‘SOLUSI-C’, yaitu solusi asimetrik, dimana filsuf pada nomor ganjil mengambil sumpit kiri dulu baru sumpit kanan, sedangkan filsuf pada nomor genap mengambil sumpit kanan dulu baru sumpit kiri.

6 : *textbox*, untuk mengisi representasi waktu 1 detik pada perangkat lunak dengan waktu sebenarnya.

7 : *textbox*, batas waktu simulasi dalam detik

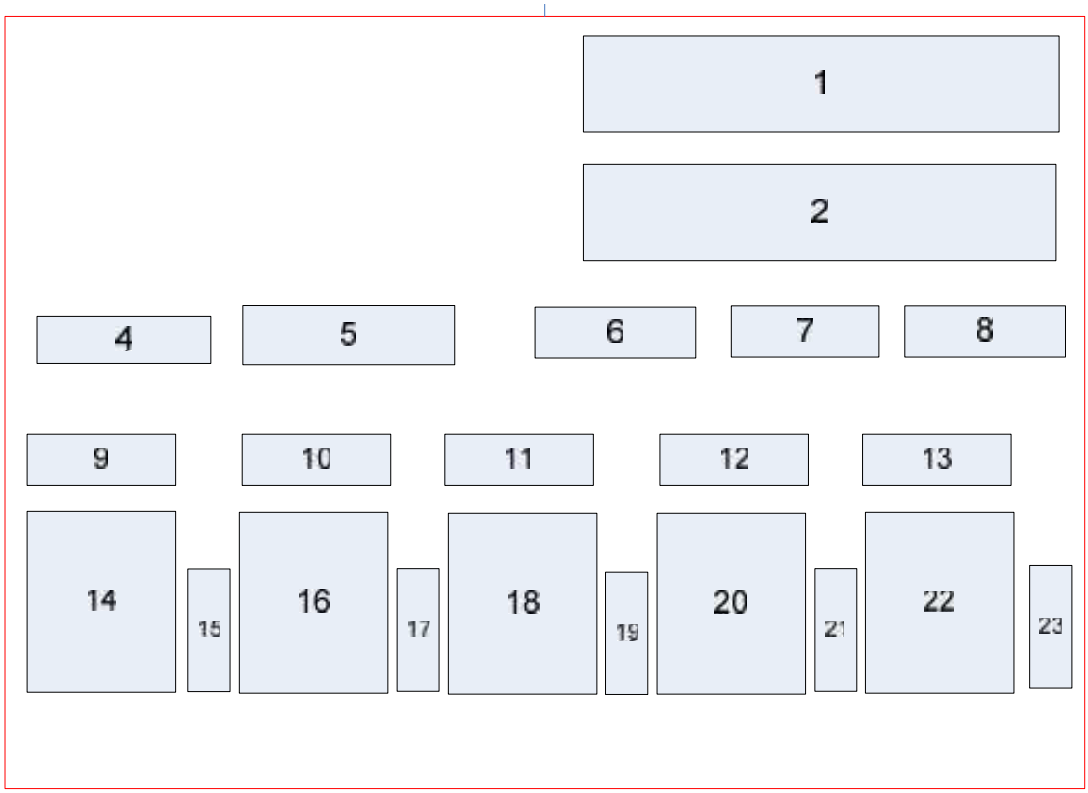
8 : tombol ‘Buka History / Log’, untuk mencatat dan menampilkan laporan proses simulasi berjalan.

9 : tombol ‘Mulai Simulasi’, untuk memulai proses simulasi.

10 : tombol ‘Keluar’, untuk menutup *form* dan keluar dari perangkat lunak.

4.2.3 Form Proses Simulasi

Form Proses Simulasi berfungsi sebagai *form* untuk mensimulasikan proses *dining philosophers problem*. Pada *form* ini, akan ditampilkan bagan *dining philosophers problem* dan visualisasi aksi yang dilakukan oleh 5 filsuf masing-masing.



Gambar 4.8 Rancangan form proses simulasi

Keterangan :

1 : tabel properti dan kondisi filsuf pada saat itu.

2 : *textbox*, sebagai tempat menampilkan proses simulasi yang terjadi.

3 : bagan *dining philosophers problem*, sebagai gambaran umum proses simulasi yang sedang terjadi.

4 : label, untuk menampilkan waktu simulasi.

5 : tombol ‘Mulai / Hentikan Simulasi’, untuk memulai, melanjutkan atau menghentikan proses simulasi.

6 : tombol ‘Keluar’, untuk keluar dari *form* proses simulasi.

7 : tombol ‘Simpan’, untuk menyimpan *history* / *log* proses simulasi.

8 : tombol ‘Cetak’, untuk mencetak *history* / *log* proses simulasi.

9 : daerah visualisasi kondisi filsuf-1 (kenyang atau lapar) dengan menggunakan *progressbar*.

10 : daerah visualisasi kondisi filsuf-2 (kenyang atau lapar) dengan menggunakan *progressbar*.

11 : daerah visualisasi kondisi filsuf-3 (kenyang atau lapar) dengan menggunakan *progressbar*.

12 : daerah visualisasi kondisi filsuf-4 (kenyang atau lapar) dengan menggunakan *progressbar*.

13 : daerah visualisasi kondisi filsuf-5 (kenyang atau lapar) dengan menggunakan *progressbar*.

14 : *picturebox*, sebagai daerah visualisasi filsuf-1.

15 : *image*, sebagai gambar dari sumpit no-1.

16 : *picturebox*, sebagai daerah visualisasi filsuf-2.

17 : *image*, sebagai gambar dari sumpit no-2.

18 : *picturebox*, sebagai daerah visualisasi filsuf-3.

19 : *image*, sebagai gambar dari sumpit no-3.

20 : *picturebox*, sebagai daerah visualisasi filsuf-4.

21 : *image*, sebagai gambar dari sumpit no-4.

22 : *picturebox*, sebagai daerah visualisasi filsuf-5.

23 : *image*, sebagai gambar dari sumpit no-5.

BAB V
IMPLEMENTASI DAN PENGUJIAN

5.1 Implementasi

Implementasi sistem program ini mencakup spesifikasi kebutuhan perangkat keras (*hardware*) dan spesifikasi perangkat lunak (*software*).

5.1.1 Spesifikasi Perangkat Keras dan Perangkat Lunak

Pada prinsipnya setiap pembuatan simulasi yang telah dirancang memerlukan sarana pendukung yaitu berupa peralatan-peralatan yang sangat berperan dalam menunjang keberhasilanya.

Komponen-komponen yang dibutuhkan antara lain *hardware*, yaitu kebutuhan perangkat keras komputer kemudian *software*, yaitu kebutuhan akan perangkat lunak berupa sistem untuk mengoperasikan sistem yang telah didesain.

Program ini dijalankan dengan menggunakan perangkat keras (*hardware*) yang direkomendasikan sebagai berikut:

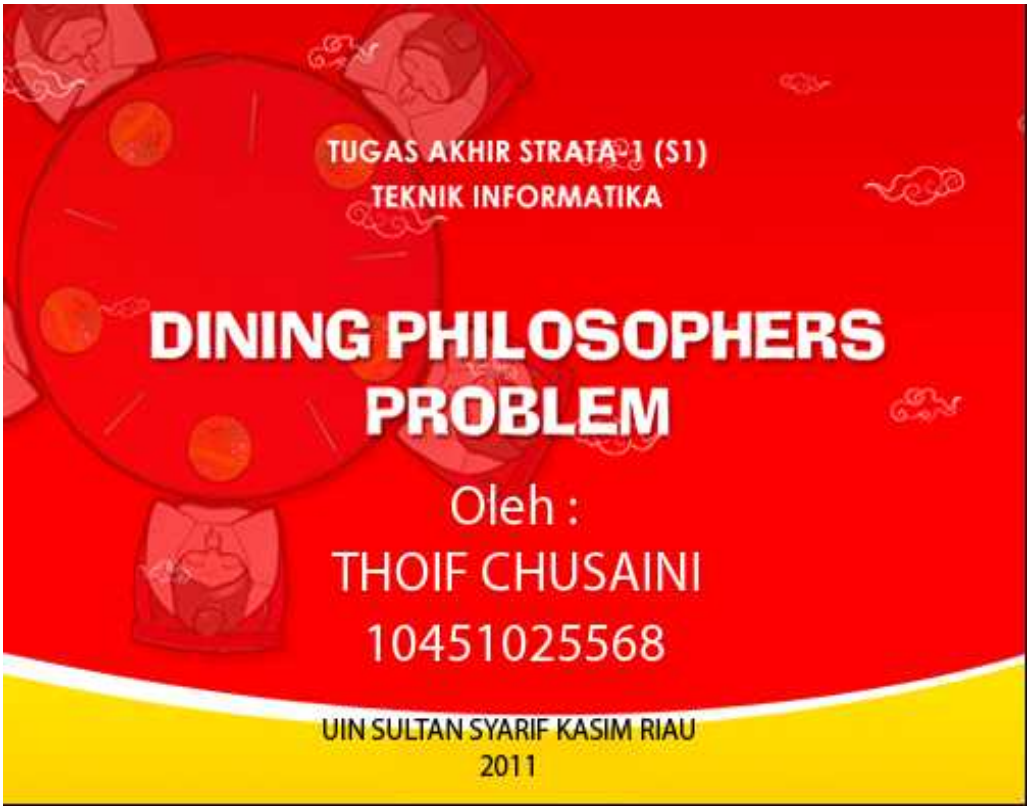
1. Prosesor Intel Pentium IV 2.26 GHz.
2. Memory 256 MB.
3. Harddisk 40 GB.
4. VGA *card* 32 MB.
5. Monitor dengan resolusi 1024 X 768 *pixel*.
6. *Keyboard* dan *Mouse*.

Sistem operasi yang direkomendasikan untuk menjalankan aplikasi ini adalah lingkungan sistem operasi Microsoft Windows/NT/2000/XP dan pemrogramannya menggunakanVisual Basic 6.0

5.1.2 Hasil Implementasi

5.1.2.1 Tampilan Splash Screen

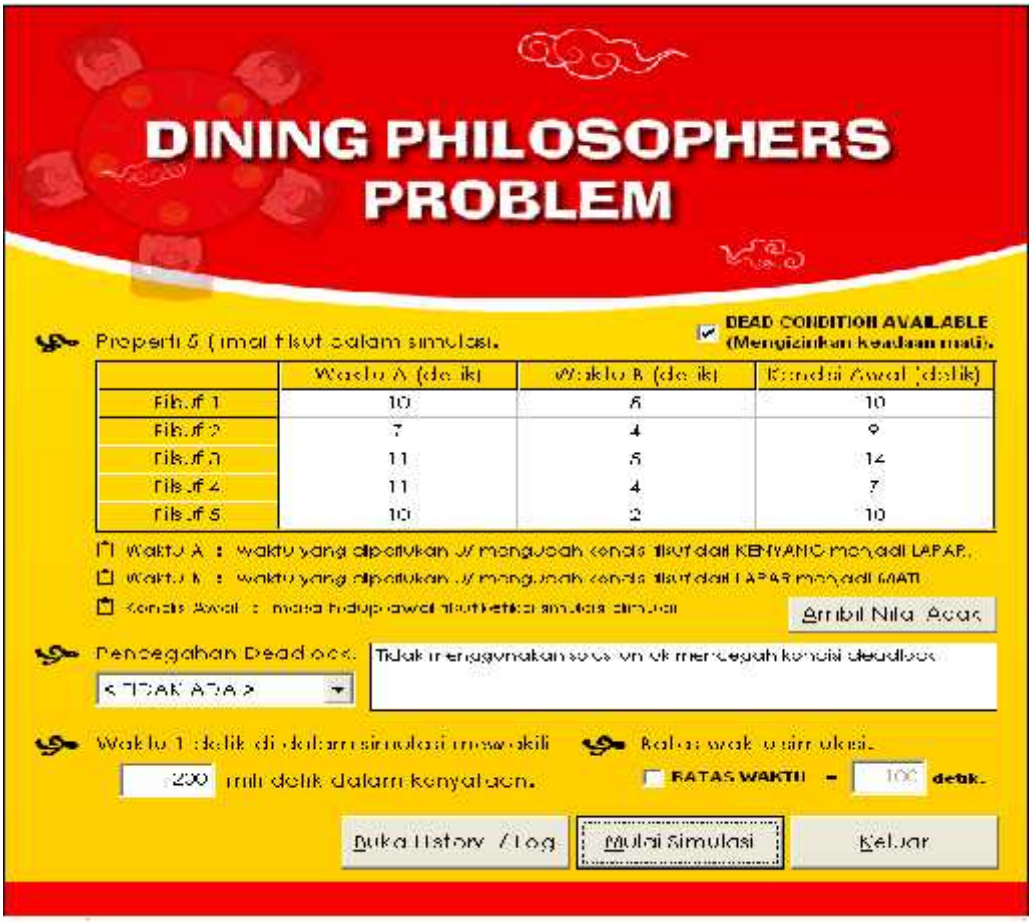
Hasil implementasi pada *form splash screen* menampilkan informasi kepada pengguna mengenai perangkat lunak yang digunakan. Informasi tersebut meliputi program studi, jurusan yang diambil, nama simulasi yang di jalankan, nama pembuat simulasi dan nama universitas.



Gambar 5.1 tampilan *splash screen*

5.1.2.2 Tampilan *Input* Data Ketika Tidak Terjadi *Deadlock*

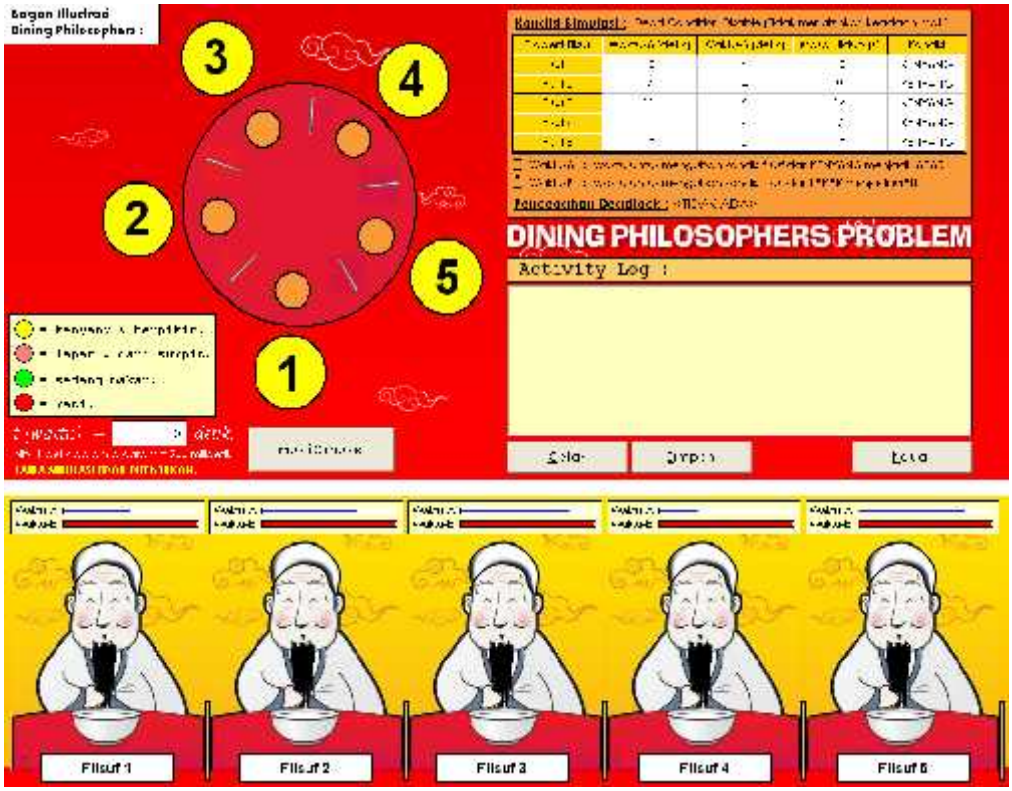
Tampilan *input* berfungsi sebagai *form* untuk memasukkan nilai perangkat lunak. Nilai perangkat lunak ini dapat diambil dari nilai acak pada komputer. Tampilan *input* data pada perangkat lunak berupa properti filsuf yang berisi waktu. Waktu ini berfungsi untuk mengubah kondisi kenyang menjadi lapar, lapar menjadi mati serta masa awal hidup filsuf. Tipe pencegahan *deadlock* ini 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 *form* ini akan dilanjutkan ke *form* proses simulasi.



Gambar 5.2 *Input* data ketka tidak terjadi *deadlock*

5.1.2.3 Tampilan Awal Simulasi Ketika Tidak Terjadi *Deadlock*

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

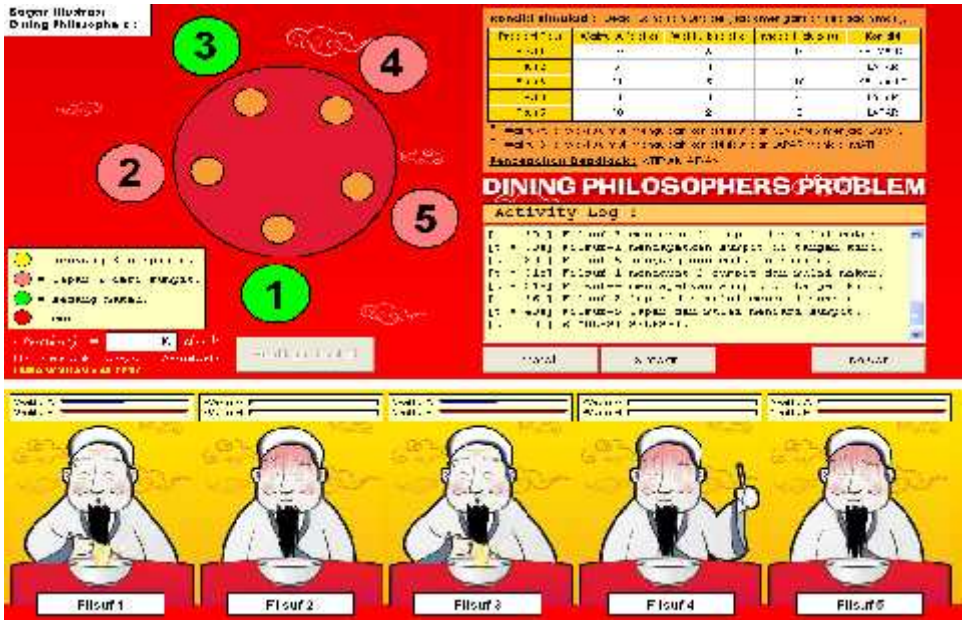


Gambar 5.3 Tampilan awal proses simulasi tidak terjadi *deadlock*

5.1.2.4 Tampilan Proses Simulasi Ketika Tidak Terjadi *Deadlock*

Tampilan proses simulasi ketika dijalankan ini menjelaskan tentang kerja dari *Dining Philosophers Problems*. Tampilan ini juga dapat menampilkan aktifitas-aktifitas para filsuf ketika dijalankan pada proses simulasi. Serta dapat langsung dicetak atau di *print* proses kerja filsuf tersebut.

Tampilan sewaktu proses simulasi sedang berjalan adalah sebagai berikut:



Gambar 5.4 Tampilan sewaktu proses simulasi sedang berjalan pada t = 40 detik

Log yang dicatat saat *t* = 49 detik adalah sebagai berikut:

DINING PHILOSOPHER LOG

[t = 3s] Filsuf-4 lapar dan mulai mencari sumpit.

[t = 3s] Filsuf-4 mendapatkan sumpit di tangan kiri.

[t = 4s] Filsuf-4 mendapat 2 sumpit dan mulai makan.

[t = 5s] Filsuf-1 lapar dan mulai mencari sumpit.

[t = 5s] Filsuf-1 mendapatkan sumpit di tangan kiri.

[t = 5s] Filsuf-2 lapar dan mulai mencari sumpit.

[t = 5s] Filsuf-2 mendapatkan sumpit di tangan kiri.

[t = 6s] Filsuf-1 mendapat 2 sumpit dan mulai makan.

[t = 8s] Filsuf-5 lapar dan mulai mencari sumpit.

[t = 9s] Filsuf-3 lapar dan mulai mencari sumpit.

[t = 16s] Filsuf-4 kenyang dan mulai berpikir.

[t = 16s] Filsuf-5 mendapatkan sumpit di tangan kanan.

[t = 17s] Filsuf-1 kenyang dan mulai berpikir.

[t = 17s] Filsuf-2 mendapat 2 sumpit dan mulai makan.

[t = 17s] Filsuf-3 mendapatkan sumpit di tangan kiri.

[t = 17s] Filsuf-5 mendapat 2 sumpit dan mulai makan.

[t = 27s] Filsuf-1 lapar dan mulai mencari sumpit.

[t = 27s] Filsuf-4 lapar dan mulai mencari sumpit.

[t = 29s] Filsuf-2 kenyang dan mulai berpikir.

[t = 29s] Filsuf-3 mendapat 2 sumpit dan mulai makan.

[t = 30s] Filsuf-1 mendapatkan sumpit di tangan kiri.

[t = 30s] Filsuf-5 kenyang dan mulai berpikir.

[t = 31s] Filsuf-1 mendapat 2 sumpit dan mulai makan.

[t = 31s] Filsuf-4 mendapatkan sumpit di tangan kiri.

[t = 36s] Filsuf-2 lapar dan mulai mencari sumpit.

[t = 40s] Filsuf-5 lapar dan mulai mencari sumpit.

[t = 45s] Filsuf-1 kenyang dan mulai berpikir.

[t = 45s] Filsuf-2 mendapatkan sumpit di tangan kanan.

[t = 45s] Filsuf-5 mendapatkan sumpit di tangan kiri.

[t = 46s] Filsuf-3 kenyang dan mulai berpikir.

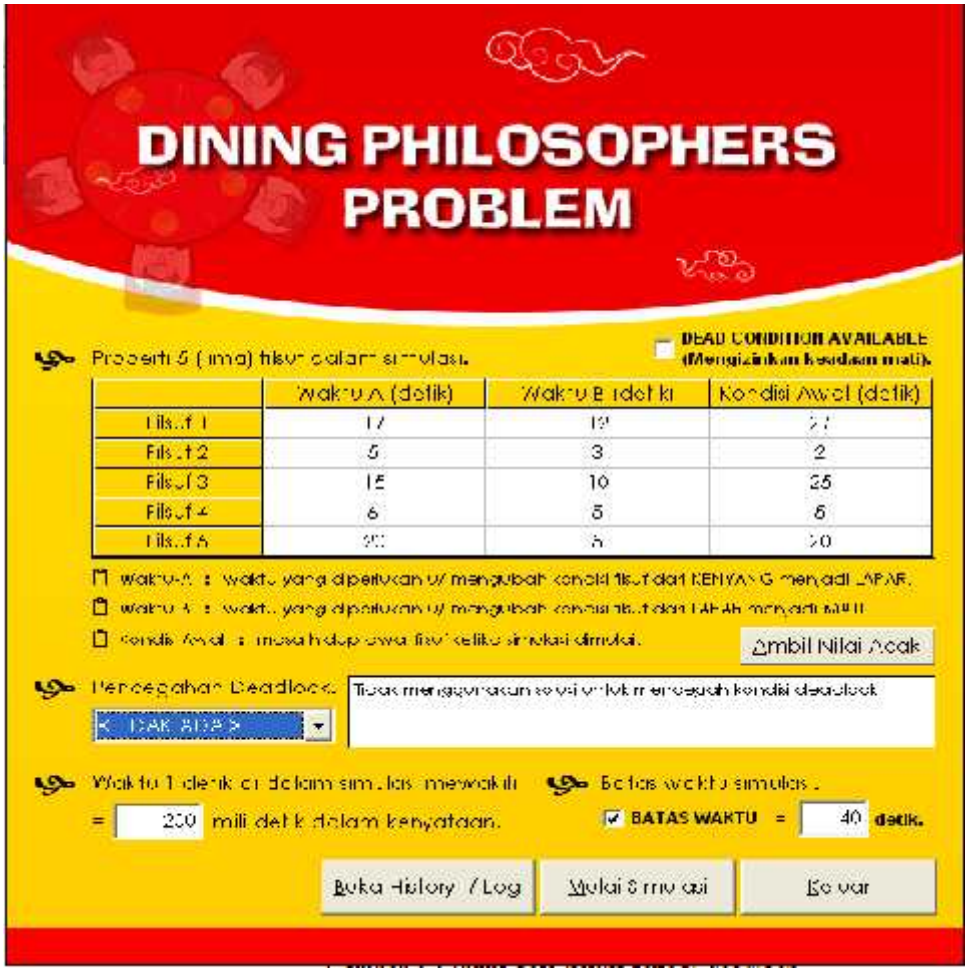
[t = 46s] Filsuf-4 mendapat 2 sumpit dan mulai makan.

[t = 47s] Filsuf-2 mendapat 2 sumpit dan mulai makan.

5.1.2.5 Tampilan *Input* Data Ketika Terjadi *Deadlock*

Tampilan *input* data ketika terjadi *deadlock* hampir sama dengan *input* data ketika tidak terjadi *deadlock*. Nilai perangkat lunak ini dapat diambil dari nilai acak pada komputer. Tampilan *input* data pada perangkat lunak berupa properti filsuf, tipe pencegahan *deadlock*, opsi '*dead condition available*' dan kecepatan simulasi.

Untuk kasus *deadlock*, perhatikan contoh berikut. *Input* data sebagai berikut:

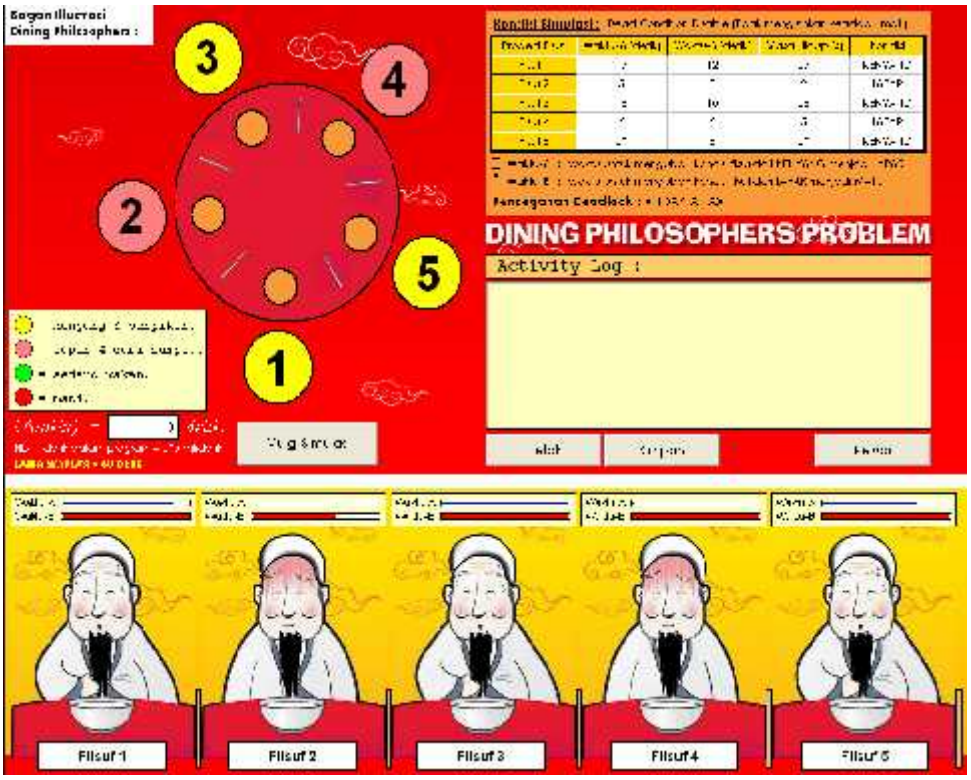


Gambar 5.5 *Input* data ketika terjadi *deadlock*

5.1.2.6 Tampilan Awal Simulasi Ketika Terjadi *Deadlock*

Tampilan awal simulasi terjadinya *deadlock* akan menampilkan nilai yang telah dimasukkan pada *form input*.

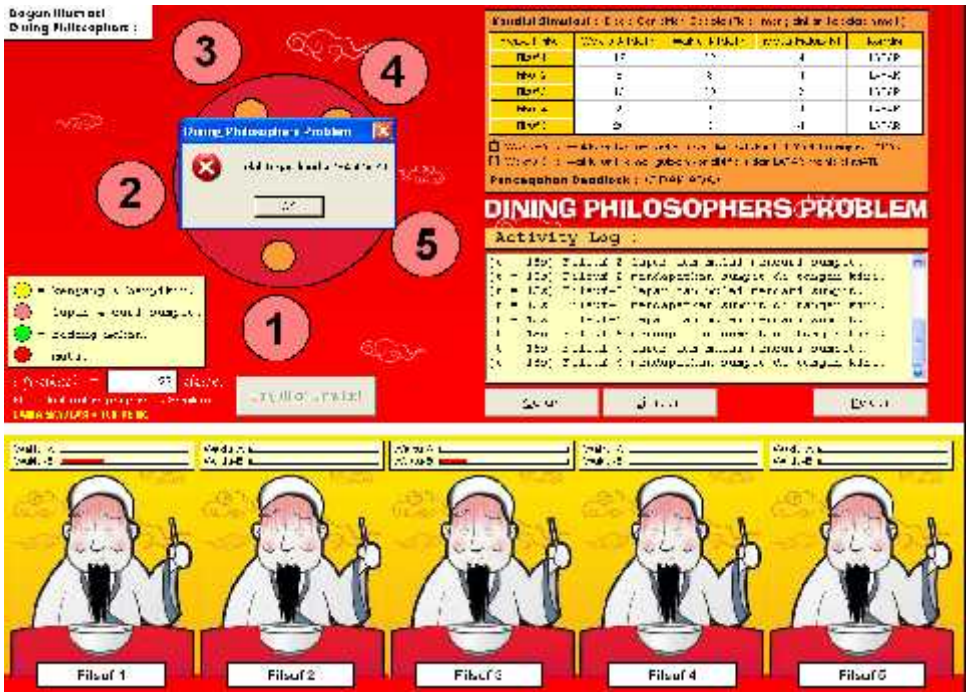
Tampilan awal simulasi ketika terjadi *deadlock* adalah sebagai berikut:



Gambar 5.6 Tampilan awal simulasi ketika terjadi *deadlock*

5.1.2.7 Tampilan Proses Simulasi Ketika Terjadi *Deadlock*

Tampilan berfungsi untuk mengetahui terjadinya *deadlock*. *Deadlock* terjadi karena pada tahap sebelumnya tidak diberikan solusi untuk pencegahannya, sehingga akan muncul tampilan sebagai berikut:



Gambar 5.7 Tampilan terjadinya kondisi *deadlock*

Log yang dicatat saat t = 23 detik adalah sebagai berikut:

DINING PHILOSOPHER LOG

[t = 1s] Filsuf-2 mendapatkan sumpit di tangan kiri.

[t = 1s] Filsuf-4 mendapatkan sumpit di tangan kiri.

[t = 2s] Filsuf-2 mendapat 2 sumpit dan mulai makan.

[t = 2s] Filsuf-4 mendapat 2 sumpit dan mulai makan.

[t = 10s] Filsuf-2 kenyang dan mulai berpikir.

[t = 10s] Filsuf-4 kenyang dan mulai berpikir.

[t = 15s] Filsuf-1 lapar dan mulai mencari sumpit.

[t = 15s] Filsuf-1 mendapatkan sumpit di tangan kiri.

[t = 15s] Filsuf-2 lapar dan mulai mencari sumpit.

[t = 15s] Filsuf-2 mendapatkan sumpit di tangan kiri.

[t = 15s] Filsuf-3 lapar dan mulai mencari sumpit.

[t = 15s] Filsuf-3 mendapatkan sumpit di tangan kiri.

[t = 15s] Filsuf-5 lapar dan mulai mencari sumpit.

[t = 15s] Filsuf-5 mendapatkan sumpit di tangan kiri.

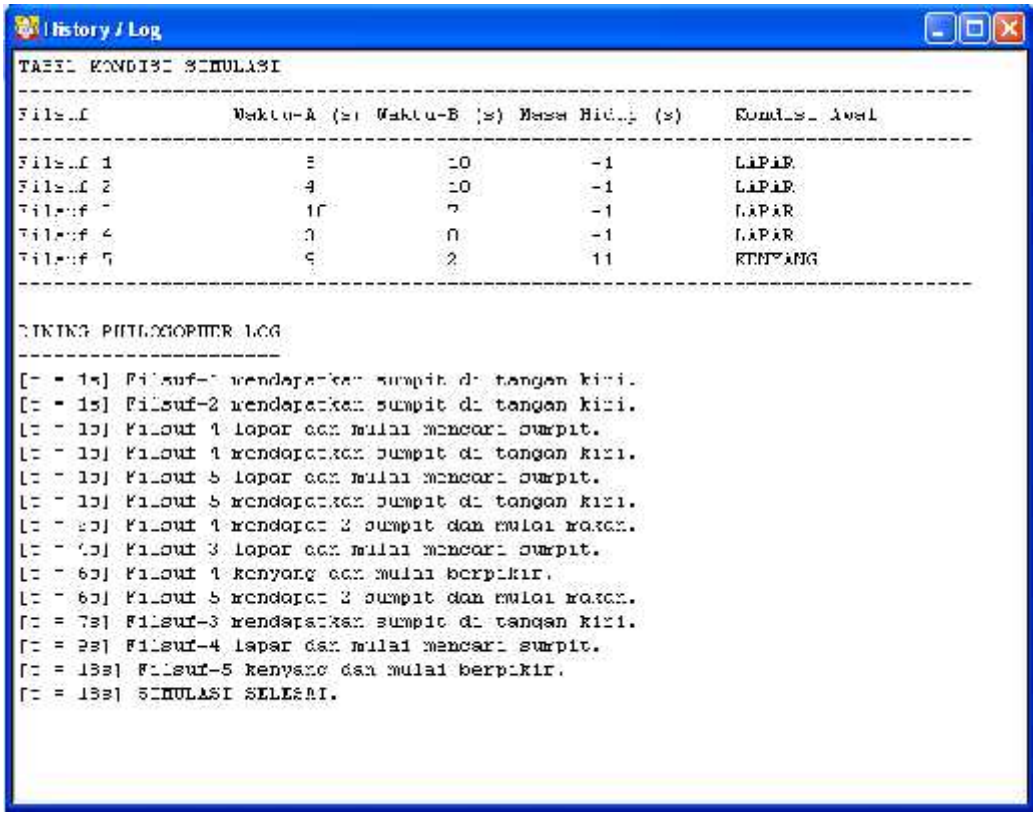
[t = 16s] Filsuf-4 lapar dan mulai mencari sumpit.

[t = 16s] Filsuf-4 mendapatkan sumpit di tangan kiri.

[t = 23s] Terjadi kondisi DEADLOCK !

5.1.2.8 Tampilan History Ketika Tidak Terjadi *Deadlock*

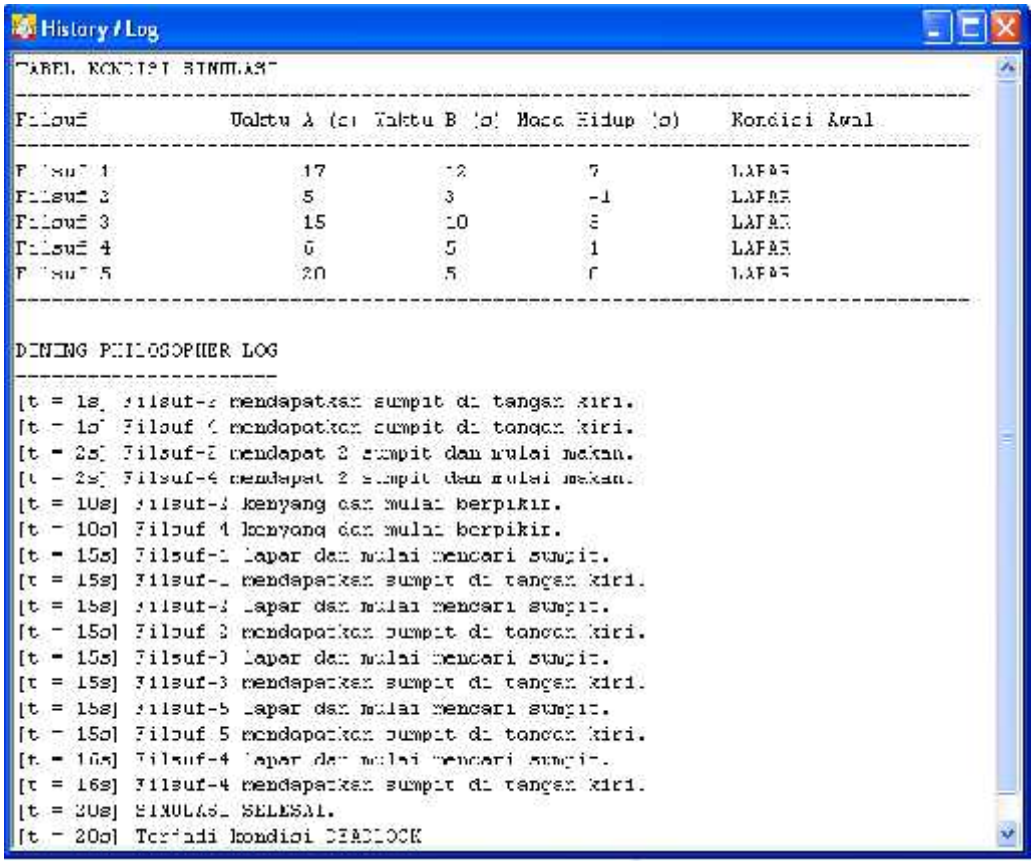
Tampilan *History* berfungsi untuk mencatat dan menampilkan laporan proses yang telah terjadi pada proses simulasi *Dining Philosopher Problems* ketika tidak terjadi *deadlock.*



Gambar 5.8 Tampilan history ketika tidak terjadi *deadlock*

5.1.2.9 Tampilan History Ketika Terjadi *Deadlock*

Tampilan *History* ketika terjadi *deadlock* berfungsi untuk mencatat dan menampilkan laporan proses yang telah terjadi pada proses simulasi *Dining Philosopher Problems* ketika terjadi *deadlock.*



Gambar 5.9 Tampilan *history* ketika terjadi *deadlock*

5.2 Pengujian Sistem (*Testing*)

Tahap *testing* dilakukan setelah selesai tahap pembuatan dan seluruh data telah dimasukkan. Suatu hal yang tidak kalah penting yaitu aplikasi harus dapat berjalan dengan baik dilingkungan pengguna. Pengguna merasakan manfaat serta kemudahan dari aplikasi tersebut dan dapat menggunakannya sendiri terutama untuk aplikasi interaktif. Pada tahap pengujian, aplikasi diuji melalui pengujian *blackbox*. Pengujian dengan menggunakan metode *blakcbox* yaitu pengujian yang dilakukan untuk antarmuka perangkat lunak, pengujian ini dilakukan untuk memperlihatkan bahwa fungsi-fungsi bekerja dengan baik atau diterima dengan benar dan keluaran yang dihasilkan benar-benar tepat, pengintegrasian eksternal data dapat berjalan dengan baik.

5.2.1 Pengujian Tampilan

5.2.1.1 Pengujian Halaman Input

Tabel 5.1 Tabel Pengujian Halaman Input

No	Deskripsi	Prekondisi	Prosedur pengujian	Masukan	Keluaran yang diharapkan	Hasil yang Didapat	Kesimpulan
1.	Pengujian Tampilan Menu Input ketika tidak terjadi *deadlock*	Pengujian Tampilan layar menu *input*	Klik menu masuk	Klik Menu masukan nilai acak	Muncul Pada layar Menu *input* tombol-tombol menu yaitu mulai simulasi, log, acak *input*, pencegahan *deadlock*, keluar	Tampilan sesuai dengan yang diharapkan	Masuk ke layar utama
				Klik Menu Acak *input*	inputan berhasil di inputkan secara acak	Tampilan sesuai dengan yang diharapkan	Masuk ke tampilan simulasi
				Klik Menu simulasi	Muncul layar tampilan simulasi	Tampilan sesuai dengan yang diharapkan	Masuk ke tampilan simulasi
				Klik Menu buka history/log	Muncul layar tampilan yang telah di simpan setelah di eksekusi	Tampilan sesuai dengan yang diharapkan	Masuk ke tampilan simulasi
2.	Pengujian keluar simulasi	Hasil tampilan layar menu utama	Klik tombol keluar	Klik menu keluar	Keluar dari simulasi *Dining Philosopher Problems*	Tampilan sesuai dengan yang diharapkan	Keluar dari simulasi

5.2.1.2 Pengujian Halaman Proses Simulasi

Tabel 5.2 Tabel Pengujian Halaman Proses Simulasi

No	Deskripsi	Prekondisi	Prosedur pengujian	Masukan	Keluaran yang diharapkan	Hasil yang Didapat	Kesimpulan
1.	Pengujian tampilan proses simulasi	Pengujian tampilan layar proses simulasi	Klik menu proses Simulasi	Klik menu proses simulasi	Muncul tampilan proses simulasi dan menu pilihan yaitu menu mulai simulasi, hentikan, lanjutkan simulasi *history/log*, tcetak, simpan, keluar	Tampilan sesuai dengan yang diharapkan	Masuk ke layar utama
				Klik menu mulai	Simulasi berjalan pada tampilan proses Simulasi	Tampilan sesuai dengan yang diharapkan	Masuk ke tampilan Simulasi
				Klik menu hentikan	Simulasi yang sedang berjalan berhenti	Tampilan sesuai dengan yang diharapkan	Masuk ke tampilan Simulasi
				Klik menu lanjutkan	Simulasi yang sedang berhenti akan berjalan kembali	Tampilan sesuai dengan yang diharapkan	Masuk ke tampilan Simulasi
				Klik menu cetak	Muncul layar tampilan perintah untuk mencetak (*print*)	Tampilan sesuai dengan yang diharapkan	Masuk ke tampilan Simulasi
				klik menu simpan	Muncul layar tampilan untuk menyimpan hasil dari proses	Tampilan sesuai dengan yang diharapkan	Masuk ke tampilan Simulasi
2.	Pengujian keluar simulasi	Hasil tampilan layar menu utama sudah tampil	Klik tombol keluar	Klik menu keluar	Keluar dari Tampilan Simulasi.	Tampilan sesuai dengan yang diharapkan	Keluar dari Tampilan Simulasi

5.2.1.3 Pengujian Tampilan Buka History/Log

Tabel 5.3 Tabel Pengujian Halaman Buka History/log

No	Deskripsi	Prekondisi	Prosedur pengujian	Masukan	Keluaran yang diharapkan	Hasil yang Didapat	Kesimpulan
1.	Pengujian Tampilan Buka History/log	Pengujian Tampilan layar Buka History/log	Klik menu Buka History/log	Klik Menu Buka History/log	Akan muncul tampilan layar untuk membuka log yang di simpan pada proses Simulasi	Tampilan sesuai dengan yang diharapkan	Masuk Ke Menu History
2.	Pengujian keluar aplikasi (*Exit*)	Hasil tampilan layar menu utama sudah tampil	Klik tombol keluar (X)	Klik menu keluar (*Exit*)	Keluar dari Tampilan History	Tampilan layar sesuai dengan yang diharapkan	keluar dari Tampilan History

5.3 Kesimpulan Pengujian

Dari hasil pengujian *Black Box*, didapatkan hasil bahwa pengujian berdasarkan *Black Box* sudah sesuai dengan *Simulasi Dining Philosophers Problems*. Keluaran yang dihasilkan oleh aplikasi ini sesuai dengan yang diharapkan yaitu berupa *Simulasi Dining Philosophers Problems* untuk mencegah terjadinya *deadlock* pada proses pengeksekusian program dengan menggunakan beberapa solusi.

BAB VI
KESIMPULAN DAN SARAN

6.1 Kesimpulan

Setelah menyelesaikan perancangan dan pembuatan simulasi pencegahan *deadlock* menggunakan *Dining Philosophers Problem* ini, penulis menarik kesimpulan bahwa *Dining Philosophers Problem* yaitu simulasi yang menggambarkan proses yang terjadi di dalam manajemen proses pada sistem operasi, dimana proses-proses saling berusaha untuk memperoleh sumber daya yang disediakan pada saat yang bersamaan, keadaan ini bisa menimbulkan problem atau yang biasa disebut kondisi *deadlock*. Kondisi *deadlock* dapat dicegah atau dihindari yaitu dengan cara menyingkirkan semua proses yang terlibat *deadlock* dan menghilangkan syarat-syarat yang dapat menimbulkan keadaan *deadlock.*

Pengujian selanjutnya adalah menggunakan pengujian *black box*, keluaran yang dihasilkan oleh aplikasi ini sesuai dengan yang diharapkan yaitu berupa simulasi pencegahan *deadlock* dengan menggunakan *Dining Philosophers Problems*.

6.2 Saran

Penulis ingin memberikan beberapa saran yang mungkin dapat membantu dalam pengembangan perangkat lunak simulasi ini yaitu :

1. Simulasi ini dapat ditambahkan fasilitas suara (*multimedia*) agar lebih menarik dalam penyajiannya.
2. Tampilan filsuf mungkin dapat ditambahkan lebih dari lima orang.
3. Perangkat lunak dapat dikembangkan dengan menggunakan beberapa *software* yang berkaitan dengan pembuatan animasi agar animasi yang dihasilkan lebih bagus dan menarik minat mahasiswa untuk memahami dan mendalami tentang sistem operasi, terutama tentang proses *deadlock* dan pencegahanya.

DAFTAR PUSTAKA

Dewi, Sri Kusuma, "Sistem Operasi Edisi Kedua", Graha Ilmu, Jakarta, 2000.

Hariyanto, Bambang, "*Sistem Operasi Edisi Kedua*", Informatika, Bandung, 2005.

Havender, J.W., "*Avoiding Deadlock in Multitasking Systems*", IBM Systems Journal, Vol.7, No.2, 1968.

Kusnadi, Kuswoyo. A, "*Sistem Operasi*", Andi, Yogyakarta, 2008.

Pangera, Abas A. "*Sistem Operasi*", Andi, Yogyakarta, 2008.

Shelly, Chashman, "*Menjelajah Dunia Komputer Fundamental*", Salemba, Infotek, 2010.

Stalling, William, "*Sistem Operasi dan Prinsip-prinsip Perancangan"*, 2003.

Stalling, William, "Sistem Operasi : Internal Dan Prinsip-Prinsip Perancangan", PT Indeks Kelompok Gramedia, Jakarta, 2006.

Wahana Komputer, "Pemrograman VB 6.0", Andi, Yogyakarta, 2000.