BAB II

TINJAUAN PUSTAKA

Pada bab ini menjelaskan mengenai tempat dilakukannya penelitian dan menjelaskan mengenai teori-teori yang akan digunakan dalam membuat sistem yang dibuat.

2.1 Sistem Terdistribusi

Sistem terdistribusi merupakan sebuah system yang terdiri dari sekumpulan mesin otomatis (*autonomous machines*) yang terhubung dengan suatu jaringan komunikasi dan dilengkapi dengan perangkat lunak (*software*) yang dirancang untuk menghasilkan suatu lingkungan komputasi yang konsisten dan terintegrasi. Sistem tedistribusi memungkinkan kita untuk saling mengkoordinasikan dan saling bekerja sama dalam melakukan aktifitas secara lebih efisien dan lebih efektif. Tujuan utama dari system terdistribusi dapat direpresentasikan dengan *: resource sharing , openness, concurrency, scalability, fault-tolerance* dan *transparency*.

a. *Resource sharing*. Dalam system terdistribusi, sumber daya (resource) – *baik hardware, software* dan data dapat digunakan oleh masing – masing user secara bersamaan. Sebagai contoh : printer dapat dipergunakan oleh sekumpulan user.
b. *Opennes*. Dalam sistem terdistribusi keterbukaan (*opennes*) dapat dicapai dengan menspesifikasikan beberapa kunci interface software

sehingga memungkinkan bagi pengembang software untuk melakukan pengembangan pada sistem tersebut.

c. *Concurrency*. Melakukan pengiriman request ke banyak mesin yang terhubung dengan jaringan secara bersamaan.

d. *Scalability*. Suatu sistem terdistribusi yang sedang berjalan pada jumlah mesin yang sedikit dapat dikembangkan (*extended*) menggunakan lebih banyak mesin untuk menghasilkan performa yang lebih baik

e. *Fault – tolerance*. Mesin- mesin yang terhubung dengan jaringan dapat dilihat sebagai suatu sumber daya yang redundant (sumber daya yang tidak diperlukan), suatu perangkat lunak sistem dapat diaplikasikan (installed) dalam banyak mesin sehingga jika terdapat suatu kesalahan pada perangkat keras maupun perangkat lunak, kesalahan tersebut dapat terdeteksi dan ditoleransi oleh mesin yang lainnya.

f. *Transparancy*. Suatu sistem yang terdistribusi dapat menyediakan banyak bentuk *transparacy*, di antaranya :

    1. *Location transparancy*, yang memungkinkan suatu informasi lokal dan *remote* dapat diakses dengan cara yang sama.
    2. *Failure transparancy*, memungkinkan pemfilteran kesalahan secara otomatis.
    3. *Replication transparancy*, memungkinkan suatu *software* atau data untuk diduplikasi pada banyak mesin secara diam2x / tidak terlihat.

2.2 Perkembangan Sistem Terdistribusi

Perkembangan teknologi sejak akhir tahun 1990 telah mencapai tahap suatu komputasi atau sistem yang berbasis web yang juga merupakan suatu sistem terdistribusi.

Dalam sistem tersebut terdapat 3 konsep dasar yaitu :

a) adanya komputer pribadi dan server komputer yang memiliki kemampuan tinggi
b) jaringan LAN dan WAN berkecepatan tinggi, internet dan
c) sistem, dalam hal sistem distribusi dan aplikasi

Dengan berkembangnya internet dan penggunaan *e-commerce*, maka sangat penting untuk merancang suatu sistem terdistribusi yang baik tidak hanya untuk aplikasi tradisional tapi juga suatu sistem terdistribusi yang diaplikasikan pada internet.

Perkembangan jaringan komputer selama 30 tahun dari tahun 1966 – 1996.

1966 *ARPA packet – switching experimentation*

1969 First Arpanet nodes operational

1972 *e-mail* terdistribusi diciptakan

1973 Komputer diluar Amerika terhubung dengan Arpanet

1975 Arpanet digunakan oleh *Defense Communication Agency*

1980 percobaan penggunaan TCP/IP dimulai

1981 Host bertambah setiap 20 hari

1983 Pengalihan penggunaan TCP/IP selesai dilaksanakan

1986 NSFnet sebagai *backbone* diciptakan

1990 Arpanet tidak dipergunakan lagi

1991 Penggunaan Gopher mulai diperkenalkan

1991 WWW diciptakan

1992 Mosaic mulai diperkenalkan

1995 Internet *backbone* mulai diprivatisasikan

1996 OC-3 (155 Mbps) backbone mulai dibangun

2.3 Arsitektur Jaringan

Dengan suksesnya ARPANET (Suatu sistem yang disponsori oleh *Advanced Research Projects Agency* (ARPA) dan dikembangkan dari tahun 1960 hingga tahun 1970) dan berbagai jaringan lainnya, dan dengan adanya potensi pertukaran paket secara langsung, satelit dan jaringan lokal akan menjadikan Jaringan Komputer sebagai area yang penting untuk dikembangkan dan dipasarkan.

Pada tahun 1978, suatu sub komite (SC16) didirikan oleh *International Organization for Standardization* (ISO) bernama *Technical Committee 97* untuk menangani masalah Pemrosesan Informasi untuk mengembangkan standar dari "*open system interconnection* (OSI) ".

Penggunaan istilah "open" di sini dipergunakan karena diharapkan sebuah sistem dapat membuka komunikasi dengan semua sistem yang ada di seluruh dunia dengan mematuhi aturan yang sama.

Model OSI yang dijadikan referensi terdiri dari model dengan tujuh lapis (layer) yang dipergunakan untuk suatu proses komunikasi yaitu *application , presentation, session, transport, network, data link* dan *physica*l.

Berikut tabel arsitektur OSI :

Tabel 2.1 Arsitektur OSI

Application
Presentation
Session
Transport
Network
Data Link
Physical

ARPANET yang dikembangkan terlebih dahulu memiliki arsitektur yang berbeda. ARPANET menggunakan arsitektur 4 lapis yang terdiri dari : *application, transport, Internet,* dan *network interface* seperti yang digambarkan pada tabel berikut :

Tabel 2.2 Arsitektur ARPANET

Application
Transport
Internet
Network Interface

2.4 *Network Fault Tolerance*

Istilah *network fault tolerance* dapat diartikan sebagai ketahanan suatu jaringan terhadap kegagalan dalam komponen jaringan tersebut.Dalam dunia bisnis saat ini, kehandalan dan ketersediaan dari suatu sistem merupakan faktor kunci. Dengan berhentinya suatu sistem dalam jangka waktu yang singkat sudah dapat mengakibatkan kerugian yang tinggi bagi bisnis yang berbasis *online*.

Pada suatu jaringan akan ditemui beberapa tantangan dalam penanganan kesalahan. Pada jaringan yang terhubung dengan internet maka akan ditemui karakteristik sebagai berikut :

1. Komunikasi yang tidak dapat diandalkan
2. Sumber daya yang tidak dapat diandalkan (komputer, media penyimpanan, perangkat lunak, dll)
3. Lingkungan yang amat heterogen (bervariasi)
4. Adanya potensi untuk bertambahnya sumber daya : scalability
5. Adanya potensi untuk bertambahnya variasi pada sumber daya.

Kehandalan dalam suatu jaringan bergantung pada ketahanan pada perangkat lunak dan perangkat keras. Kegagalan dalam perangkat lunak dapat terjadi karena adanya suatu kesalahan *protocol*. Suatu teknik dasar digunakan untuk mengatasi kegagalan dalam suatu jaringan; termasuk *retry* (*retransmission*), *complemented retry* dengan koreksi, replikasi, *coding*, protokol istimewa *(single handshake, double handshake), timing checks, rerouting,* dan *intelligent retry*.

2.5 Protokol dan QoS (*Quality of Service*)

Suatu aplikasi jaringan akan dibagi – bagi ke dalam suatu lapisan hirarki. Antar tiap lapisan tersebut akan terjadi komunikasi yang diatur oleh suatu protokol. Protokol secara garis besar merupakan suatu kesepakatan antar pihak yang melakukan komunikasi yang mengatur bagaimana komunikasi itu dilakukan. Definisi dari suatu protokol mengandung dua poin penting, yaitu :

1. Suatu spesifikasi urutan dari rentetan pesan yang akan dipertukarkan
2. Suatu spesifikasi format data yang terdapat dalam pesan.

Quality of Service (QoS) meruapakn suatu istilah yang digunakan untuk menggambarkan teknologi yang mengklasifikasikan teknologi jaringan yang digunakan.

2.6 Perangkat Lunak untuk Komputasi yang Terdistribusi

Ada beberapa model perangkat lunak yang dapat diterapkan untuk membuat suatu sistem terdistribusi yang terkomputasi. Dalam sub bab ini akan dijelaskan secara umum mengenai 2 model yang sering digunakan.

2.6.1 Model Tradisional *Client – Server*

Model client server sudah mendominasi untuk penggunaan sistem terdistribusi sejak tahun 1980. Perkembangan model ini pada akhirnya memungkinkan untuk user bekerja pada komputer masing – masing yang terhubung dengan jaringan lokal (LAN). Inti dari model ini adalah

bagaimana cara mengakses, menggunakan dan membagi (share) suatu sumber daya (*resource*) yang terdapat pada komputer lain. Pada tahun 1980, komputer dikendalikan oleh sebuah sistem operasi yang memiliki kernel *monolithic* , dimana semua servis merupakan bagian dari keseluruhan perangkat lunak. Pada tahun 1990 model *client – server* sudah dapat digunakan untuk mengembangkan banyak aplikasi. Hal ini dikarenakan model tersebut akan mengurangi biaya baik dalam pengimplementasian dan biaya perawatan.

2.6.2 Model Distribusi Berbasis *Web*

Internet dan WWW memiliki dampak yang besar dalam pengembangan komputasi terdistribusi.Untuk memenuhi kebutuhan akan cepatnya perkembangan Internet, komputasi terdistribusi perlu berpindah dari lingkungan LAN ke internet.

Pada level eksekusi, suatu sistem terdistribusi akan bergantung pada beberapa hal, diantaranya :

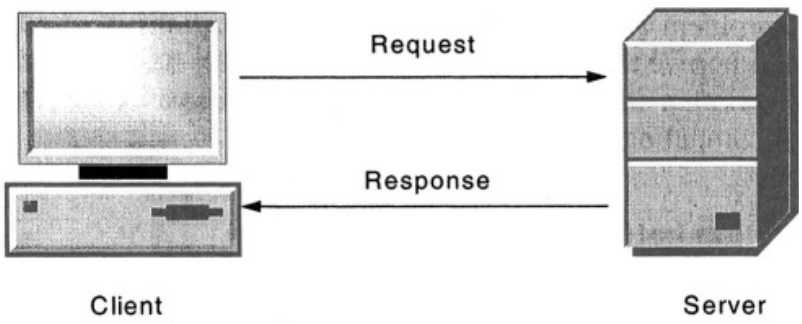
1. *Processes* : pada umumnya suatu sistem operasi dalam sebuah komputer dapat menjalankan beberapa proses secara bersamaan. Sebuah proses diciptakan dengan menjabarkan sekumpulan langkah dalam bahasa pemrograman, yang nantinya akan dikompilasi untuk dijalankan pada sistem operasi tertentu. Pada saat dijalankan, sebuah proses memiliki akses ke sumber daya pada komputer melalui sistem operasi. Sebuah proses dapat seluruhnya

didedikasikan untuk sebuah aplikasi atau dipergunakan oleh banyak aplikasi.

2. *Threads* : Setiap proses setidaknya memiliki satu *thread*. Beberapa sistem operasi mendukung penggunaan banyak *thread* dalam sebuah proses. Setiap *thread* dalam sebuah proses dapat berjalan secara terpisah dari thread lainnya. Pada umumnya *thread* memerlukan sinkronisasi. Sebagai contoh, sebuah *thread* dipergunakan untuk memonitor input dari sebuah *socket* koneksi, sedangkan proses mengakses *database*.
3. *Distributed Objects* : Dengan pendekatan ini, interaksi kendali antar komponen terjadi seluruhnya pada objek yang diminta – suatu metode eksplisit yang memanggil fungsi yang sudah tertanam sebelumnya.
4. *Agents* : Merupakan suatu entitas aktif yang bertujuan untuk mengikuti peraturan tertentu.

2.7. Konsep Dasar *Client – Server*

Pada konsep ini terdapat dua buah proses, *client* , yang meminta (request) sebuah servis dari proses yang lain, dan server, yang merupakan penyedia servis. Server akan melakukan kegiatan yang diminta dan mengirim umpan balik sebagai respons. Respons ini dapat berupa hasil, konfirmasi, maupun peringatan tentang kegagalan dari suatu operasi.



Gambar 2.1 Hubungan antara *client* dan server

Sebuah model *client server* yang lebih terperinci memiliki 3 komponen :

1. *Service* : merupakan sebuah entitas perangkat lunak yang berjalan pada satu atau lebih mesin. *Servic*e menyediakan suatu abstraksi dari retetan operasi yang diminta oleh perangkat lunak.
2. *Server* : sebuah *server* merupakan sekumpulan *service* yang berjalan pada sebuah mesin.
3. Client : sebuah entitas perangkat lunak yang mengeksploitasi *service* yang disediakan oleh *server*. Sebuah *client* dapat berinteraksi dengan user secara langsung, tetapi tidak harus.

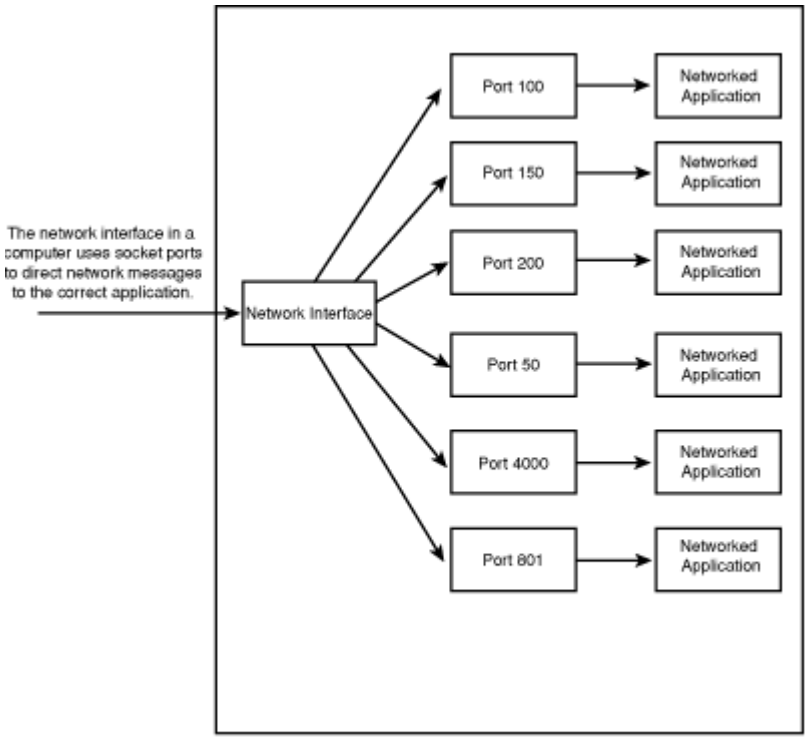
Ada 3 masalah utama yang terdapat pada model *client – server* :

1. Jika komputer yang menjadi *server* mengalami gangguan sehingga tidak dapat beroperasi sebagaimana mestinya, maka servis yang diberikan oleh komputer tersebut pun tidak dapat dijalankan. Oleh karena itu, *reliability* dan *availability* dari suatu operasi yang bergantung pada banyak server merupakan suatu produk yang bergantung pada perangkat keras dan jaringan komunikasi.

2. Permasalahan kedua yaitu adanya kemungkinan timbul potensi *bottleneck.* Hal ini terjadi karena penambahan *client* pada sistem.
3. Permasalahan terakhir timbul pada saat mengimplementasikan banyak fungsi yang serupa pada suatu sistem untuk menjaga performansi dan konsistensi. Hal ini meningkatkan biaya keseluruhan dari sistem distribusi tersebut.

2.8 *Port*

Port merupakan suatu alamat gerbang dimana suatu aplikasi ditentukan. Dalam suatu komputer terdapat banyak *port* yang tersedia, namun tidak semua *port* tersebut akan digunakan dalam satu koneksi. Hanya satu alamat port yang umumnya digunakan oleh setiap aplikasi jaringan. Hal ini dapat digambarkan sebagai berikut :



Gambar 2.2 Gambaran umum port

Pada umumnya sebuah *port* menunjuk pada sebuah aplikasi.

2.9. Sinkronisasi dalam Sistem Terdistribusi

Salah satu hal yang penting dalam sistem distribusi selain komunikasi data adalah bagaimana suatu komponen dalam sistem terdistribusi dapat bekerja sama dan saling bersinkronisasi antara satu dengan yang lainnya. Berikut akan dijelaskan beberapa sinkronisasi pada suatu sistem terdistribusi menurut Springer.

2.9.1. *Clock Synchronization*

Dalam suatu sistem terdistribusi akan sulit untuk mengumpulkan semua informasi ke tentang sistem dalam satu tempat, dan kemudian membiarkan suatu process untuk memeriksa keseluruhan sistem untuk

membuat suatu keputusan. Secara umum, suatu sistem terdistribusi memiliki kriteria – kriteria berikut :

1. Informasi yang relevan tersebar di beberapa mesin
2. Keputusan dibuat berdasarkan pada informasi lokal
3. Suatu kesalahan di satu titik sedapat mungkin harus dihindari
4. Tidak adanya sumber informasi waktu yang sangat tepat

Tiga point pertama menyatakan bahwa tidak diperkenankan untuk mengumpulkan semua informasi ke dalam satu tempat untuk diproses. Sebagai contoh, untuk melakukan penanganan sumber daya (resource), secara umum tidak diperkenankan untuk mengirim semua request ke dalam satu proses manager, yang nantinya akan memeriksa semua permintaan (request) dan mengeksekusikan serta menolak request berdasarkan informasi yang terdapat pada tabel mesin tunggal tersebut. Dalam sebuah sistem yang besar, solusi tersebut akan sangat membebani proses tersebut.

Secara ideal sebuah sistem yang terdistribusi diharapkan lebih dapat diandalkan dibandingkan dengan mesin individual. Jika satu mati, maka sisanya diharapkan dapat tetap berjalan. Kegagalan pada satu mesin yang kemudian menghentikan keseluruhan sistem merupakan hal yang paling dihindari pada sistem terdistribusi.

Point terakhir pada daftar di atas merupakan hal yang penting karena dalam suatu sistem terdistribusi, mencapai suatu kesepakatan dalam waktu bukan hal yang boleh dikesampingkan. Pada saat suatu proses A meminta waktu, proses tersebut akan memanggil sistem dan kernel menjawabnya. Jika

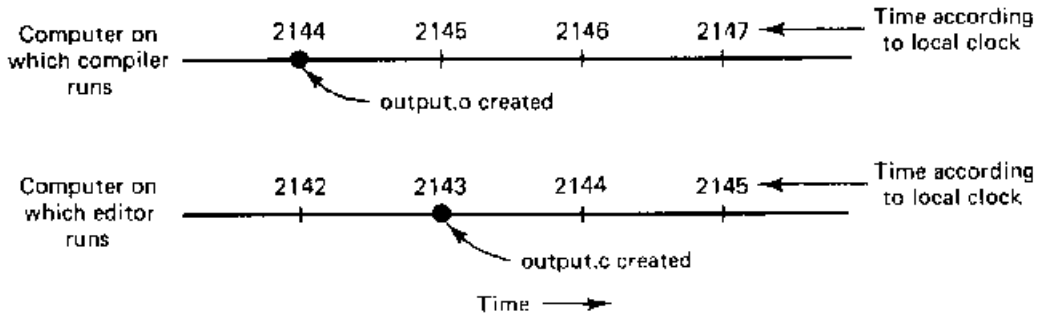
proses A meminta waktu, dan kemuidian setelah beberapa saat proses B meminta waktu, nilai waktu yang didapat oleh B akan lebih tinggi (atau memungkinkan sama) dibandingkan nilai yang didapat oleh A. Nilai yang didapat tentunya tidak lebih rendah.

Sebagai contoh dapat diambil dari perintah *make* dari UNIX. Pada UNIX, suatu program dipecah menjadi banyak *source file* sehingga perubahan pada suatu *source file* hanyak membutuhkan satu file untuk dikompilasi, tidak semua file.

Perintah *make* pada dasarnya prinsip kerjanya sederhana. Pada saat programmer menyelesaikan perubahan pada *source file*, maka dia akan memulai *make*_, yang kemudian memeriksa waktu pada semua source file dan object file dimana file tersebut terakhir dimodifikasi. Jika source file input.c memiliki waktu 2151 dan object yang berkorespondensi dengan file tersebut input.o memiliki waktu 2150, maka *make* akan mengetahui bahwa file input.c sudah berubah setelah file input.o dibuat. Maka dari itu file input.c harus dikompilasi kembali. Sebaliknya, jika pada saat suatu file output.c memiliki waktu 2144 dan object output.o memiliki waktu 2145 maka tidak akan dilakukan suatu kompilasi.

Dapat dibayangkan jika pada suatu sistem terdistribusi yang tidak terdapat suatu kesepakatan global terhadap waktu. Misalkan suatu output.o memiliki waktu 2144 dan beberapa saat kemudian output.c dimodifikasi tetapi memiliki waktu 2143 karena *clock* pada mesin tersebut sedikit lebih lambat, maka perintah *make* tidak akan memanggil *compiler*. Hasilnya,

program biner yang dapat dieksekusi akan memiliki campuran antara file object yang didapat dari source code terdahulu dan terbaru. Kemungkinan program tersebut tidak dapat berjalan dan membuat programmer akan sulit memahami apa yang salah dengan source code tersebut.



Gambar 2.3 Skema pemrosesan berdasarkan waktu

2.9.2. *Logical Clocks*

Hampir seluruh komputer memiliki sebuah *circuit* untuk menunjukkan waktu. Pada kenyataannya *circuit* tersebut bukanlah penunjuk waktu (jam) yang sebenarnya. Kata yang tepat untuk mendeskripsikan *circuit* tersebut adalah *timer*. Timer pada suatu komputer pada umumnya merupakan suatu crystal quartz yang termekanisasi. Jika dihadapkan pada suatu tekanan, kristal tersebut akan berosilasi pada frekuensi tertentu bergantung pada jenis kristal dan bagaimana kristal tersebut dipotong serta seberapa besar tekanan yang diberikan. Terdapat 2 register yang berasosiasi dengan kristal tersebut. Sebuah counter dan holding register. Setiap osilasi yang terjadi akan mengurangi counter jumlah counter. Jika counter mencapai nol, maka suatu

interript akan diregenerasi dan counter akan kembali terisi oleh nilai yang terdapat pada holding register. Dengan begini sangat memungkinkan untuk memrogram sebuah timer untuk meregenerasi 60 interrupt tiap detiknya atau sesuai dengan frekuensi yang diinginkan. Setiap interrupt disebut dengan satu clock tick.

Pada saat sistem boot pertama kali, maka sistem akan meminta suatu operator untuk memasukan tanggal dan waktu yang kemudian dikonversikan dengan jumlah detak (ticks) yang terjadi setelah waktu dan tanggal yang tersimpan dalam memori. Dalam setiap detak yang terjadi, interrupt service akan menambah waktu pada memory yang tersimpan.

Untuk mensinkronisasikan logical clock. Lamport mendefinisikan sebuah relasi yang bernama happens – before. Suatu expresi a -> b dapat dibaca “a happens before b” a terjadi sebelum b. Secara umum suatu relasi happens – before dapat diamati dalam dua situasi :

1. Jika a dan b merupakan suatu event dalam sebuah proses yang sama , dan a terjadi sebelum b maka a->b adalah true
2. Jika a merupakan suatu event yang dikirim oleh satu event proses dan b merupakan suatu event yang diterima oleh proses yang lain, maka a->b adalah true. Suatu message tidak dapat diterima sebelum dikirimkan terlebih dahulu, atau diterima dan dikirimkan secara bersamaan karena dibutuhkan waktu untuk sebuah pesan tiba.

Happens – before merupakan suatu relasi yang transisitve, jadi apabila a->b dan b->c maka a->c. Jika dua events x dan y terjadi pada proses yang tidak memerlukan pertukaran message, maka x->y tidak benar dan juga y->x. Event tersebut dikatakan concurrent.

Apabila yang dibutuhkan adalah pengukuran waktu untuk tiap event, a, maka dapat ditentukan suatu nilai waktu C(a) yang nantinya akan disepakati oleh semua proses yang akan berlangsung.

Nilai C harus terus bertambah (maju) dan tidak pernah berkurang (mundur). Pembenaran pada waktu bisa dengan penambahan nilai positif tetapi tidak dengan pengurangan nilai.

Dengan prinsip tersebut maka terdapat suatu cara untuk mengalokasikan waktu pada semua event yang terjadi dalam sistem terdistribusi.

1. Jika a terjadi sebelum b dalam suatu proses yang sama maka C(a) < C(b).
2. Jika a dan b mewakili pengiriman dan penerimaan message, maka C(a) < C(b)
3. untuk semua event a dan b, C(a) != C(b).

2.9.3. *Physical Clocks*

Pada beberapa sistem (real time sistem), waktu sesungguhnya akan sangat penting untuk diketahui. Untuk sistem tersebut waktu eksternal dibutuhkan. Dalam hal ini terdapat suatu sistem yang disebut dengan Universal Coordinated Time yang disingkat UTC. UTC merupakan basis

penentu waktu peradaban modern yang menggantikan standard lama Greenwich Mean Time.

2.9.4. *Clock Synchronization Altorithms*

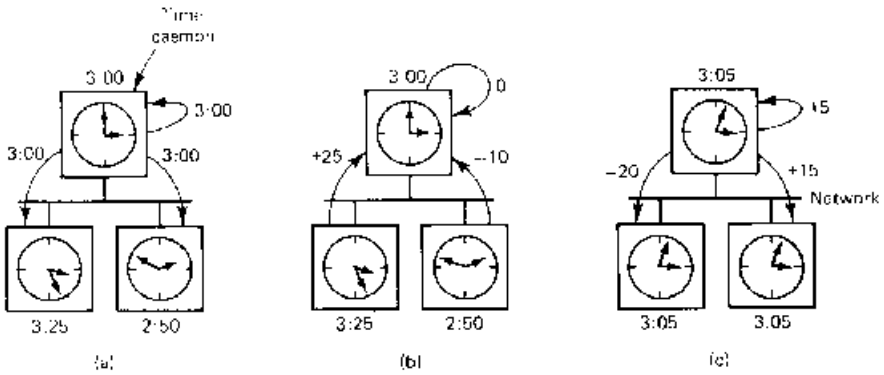
Jika sebuah mesin memiliki WWV receiver, maka mesin yang lain akan mensinkronkan dengan WWV receiver tersebut. Jika tidak ada mesin yang memiliki WWV receiver maka tiap mesin sedapat mungkin akan menyamakan waktu dengan semua masin yang ada. Terdapat beberapa algoritma dalam mensinkronisasikan antara satu mesin dengan mesin yang lain.

2.9.4.1. Algoritma Christian

Suatu algoritma yang menganggap satu mesin sebagai time server. Semua mesin akan menanyakan waktu yang tertera pada time server tersebut. Mesin tersebut (time server) kemudian meresponse dengan waktu yang sesingkat – singkatnya dengan mengirimkan pesan berisi waktu pada saat itu (C utc). Pada saat pengirim pesan menerima jawaban (reply) maka mesin tersebut dapat menset waktu menjadi C utc. Namun algoritma ini memiliki 2 kelemahan, satu minor dan satu mayor. Kelemahan mayor nya adalah waktu tidak boleh berjalan mundur. Apabila pengirim pesan memiliki waktu lebih cepat maka Cutc akan bernilai lebih kecil. Kelemahan berikutnya adalah dibutuhkan waktu untuk mengirimkan balasan dari time server.

2.9.4.2. Algoritma Berkeley

Pada algoritma Cristian, time server pasif, yang dilakukan adalah meresponse pada query yang diberikan. Pada algoritma berkely time server aktif, time server akan melakukan polling pada setiap mesin secara berkala dan menanyakan waktu pada tiap mesin. Berdasarkan jawaban yang diberikan, time server mengkomputasi dan memberikan perintah untuk mempercepat atau memperlambat waktu. Untuk lebih jelasnya dapat dilihat pada gambar berikut :



Gambar 2.4 Diagram sinkronisasi waktu algoritma Berkeley

2.9.5. Mutual Exclusion

Sistem yang melibatkan banyak proses merupakan suatu yang dengan mudah diprogram pada region critical. Bila suatu proses harus membaca atau mengupdate suatu data yang di*share*, maka proses tersebut memasuki critical region untuk mencapai mutual exclusion dan memastikan tidak ada proses yang menggunakan data yang di share tersebut pada waktu yang bersamaan. Hal ini dapat diimplementasikan pada sistem terdistribusi.

Menurut Nancy model mutual exclusion pertama dikembangkan pada tahun 1965 oleh Edsger Djikstra. Variabel *flag*[*i*] merupakan variabel yang hanya dapat ditulis atau diubah oleh satu proses tapi dapat dibaca oleh semua atau banyak proses. Variabel *turn* merupakan suatu variabel yang dapat ditulis atau diubah dan dibaca oleh banyak proses.

Algoritma Djikstra menjamin terjadinya suatu mutual exclusion, sehingga pada saat menjalankan banyak thread sekaligus, suatu resource dapat dijaga agar hanya dapat diakses oleh satu proses. Berikut adalah algoritma mutual exclusion yang dikemukakan oleh Djikstra :

```
  ** Remainder Region **
rrg_i
  ** begin first stage, trying to obtain turn **
L: flag[i] ← 1
while turn ≠ i do
    if flag[turn] = 0 then turn ← i
    end if
end while
  ** begin second stage, checking that no other processor has reached this stage **
flag[i] ← 2
for j ≠ i do ** Note: order of checks unimportant **
    if flag[j] = 2 then goto L
    end if
end for
crit_i
  ** Critical Region **
exit_i
flag[i] ← 0
rem_i
```

Keterangan :

flag[i] merupakan suatu variabel array integer yang memiliki nilai {0,1,2} yang dapat dibaca oleh semua proses tapi hanya dapat diubah (ditulis) oleh satu proses.

turn merupakan suatu variabel integer yang dapat dibaca dan diubah (ditulis) oleh banyak proses.

2.9.6. *Centralized Algorithm*

Cara yang paling umum diguinakan untuk mencapai mutual exclusion pada sistem terdistribusi adalah dengan cara menstimulasikan bagaimana mutual exclusion dapat terjadi pada sebuah sistem dengan procesor tunggal. Sebuah proses akan terpilih sebagai koordinator. Apabila ada sebuah proses yang meminta akses ke critical region maka proses tersebut mengirim request ke coordinator dengan menyebutkan critical region yang diinginkan. Jika tidak ada proses lain yang terdapat pada critical region, maka coordinator mengirim reply yang menyatakan permintaan akses ke critical region dberikan.

Coordinator hanya akan memperbolehkan satu proses dalam satu waktu untuk mencapai mutual exclusion. Request akan diperiksa berdasarkan urutan kedatangan.

2.9.7. *Distributed Algorithm*

Jika suatu proses ingin memasuki critical region, maka dia akan membuat suatu message yang berisi nama dari critical region yang ingin diakses tersebut, nomor proses, dan waktu saat itu. Kemudian proses tersebut akan mengirimkan ke proses yang lain, termasuk pada proses itu sendiri. Kemudian apabila suatu proses menerima message request dari proses lain, aksi yang dilakukan tergantung pada critical region yang terdapat pada pesan. Tiga kasus yang dapat terjadi adalah sebagai berikut :

1. Apabila penerima tidak terdapat pada critical region dan tidak ingin memasuki ciritcal region tersebut maka proses tersebut akan mengirim message OK ke pengirim.
2. Apabila penerima sudah terdapat pada critical region, maka tidak akan memberi balasan, tetapi mengantrikan request tersebut.
3. Jika penerima ingin memasuki critical region tetapi belum melakukannya, maka akan dilakukan perbandingan waktu dengan waktu yang tertera pada message request yang diterimanya. Waktu yang terkecil yang akan didahulukan. Jika pesan yang diterima memiliki time stamp yang lebih rendah maka penerima akan mengirim pesan OK. Jika pesan proses itu sendiri yang lebih kecil, maka penerima akan mengantrikan request tersebut dan tidak mengirimkan balasan.

Setelah mengirm request untuk meminta akses ke critical region, proses tersebut akan menunggu jawaban dari seluruh proses.Apabila semua hak akses telah diterima, maka proses tersebut akan memasuki critical region.

Setelah keluar dari critcal region maka proses tersebut akan mengirim message OK kepada semua proses dan menghapus semua proses tersebut dari antrian.

2.10. *Multithreading*

Pada sebuah *processor*, *multithreading* secara umum berlangsung dengan cara time-division *multiplexing* (multitasking) : sebuah processor berpindah dari satu *thread* ke *thread* yang lain. Hal tersebut terjadi secara cepat sehingga *user* akan beranggapan bahwa suatu thread atau task berjalan secara bersamaan. Pada suatu multiprocessor atau *multi-core system*, suatu *thread* akan dijalankan bersamaan, dengan setiap *processor* atau *core* menjalankan sebuah *thread* atau task. Dukungan *thread* dalam bahasa pemrograman bervariasi : tidak semua memibahasa pemrograman mendukung lebih dari satu eksekusi dalam sebuah program untuk dijalankan secara bersamaan.

Perbedaan dasar dari *threads* dan *processes* adalah :

1. *Processes* pada umumnya independen, sementara threads merupakan subset dari sebuah process
2. *Processes* memiliki alamat berbeda, sedangkan thread saling membagi alamat memori.
3. *Context switching* antara *thread* pada process yang sama cenderung lebih cepat dibandingkan context switching antara berbeapa process.

Dalam banyak situasi, suatu aplikasi dapat melakukan pekerjaan yang hamper sama oleh banyak client. Akan dapat terjadi suatu situasi seperti berikut :

1. Sebuah *web serve*r menerima banyak request dari *client* untuk menampilkan suatu web page.
2. Jika suatu server menggunakan teknologi tradisional (single – threaded) maka web server hanya akan dapat melakukan satu pelayanan request dalam waktu yang bersamaan.

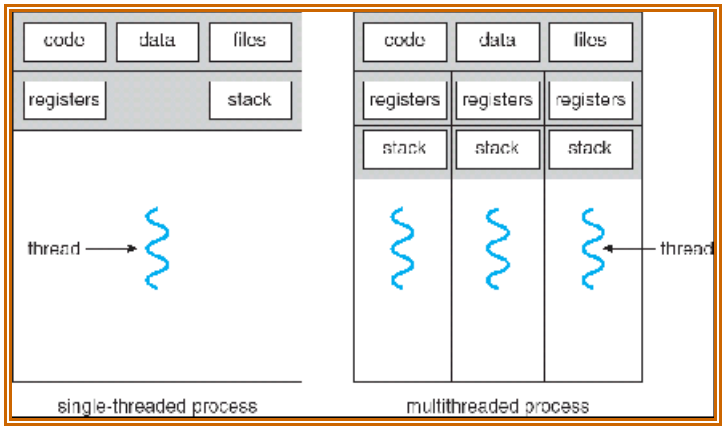
Ada beberapa pendeketan yang dapat diambil untuk mengatasi masalah tersebut. Dengan pendekatan multi – threading kita dapat melakukan hal – hal sebagai berikut :

Solusi pertama :

a. Jika seuatu server menerima sebuah request, server tersebut akan membuat suatu proses yang terpisah yang nantinya akan melakukan request tersebut.
b. Proses ini sangat memakan waktu dan menggunakan resource yang cukup besar

Solusi kedua :

Jika suatu request dibuat, suatu server akan menciptakan suatu thread lain untuk melayani request tersebut.

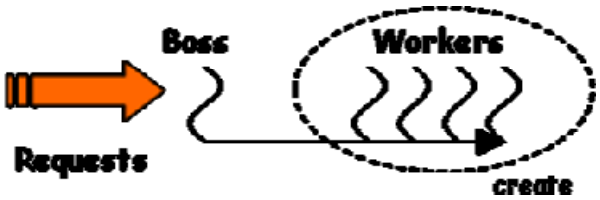


Gambar 2.5 Perbandingan *single thread* dan *multithread*

2.10.1 Model *boss worker* thread untuk suatu server *multithreading*

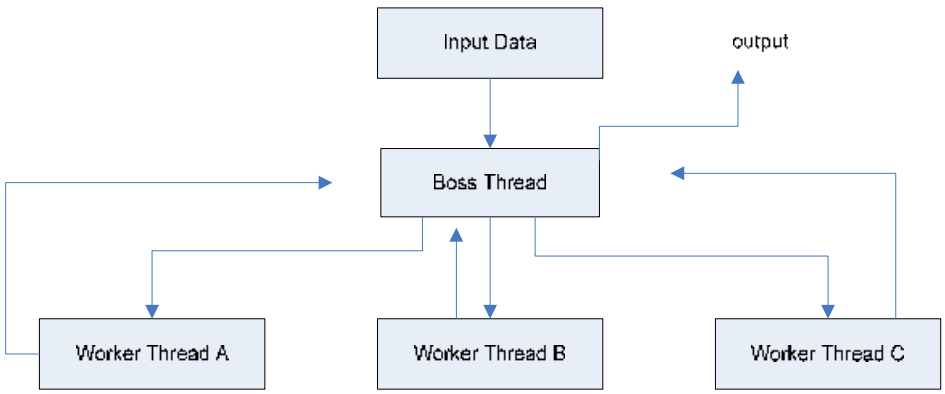
Boss / Worker model merupakan suatu model yang umum digunakan untuk server yang menggunakan konsep multithreading. Sesuai dengan namanya, model Boss – Worker terdiri dari 2 bagian. Pada prinsipnya bagian pertama yaitu Boss, merupakan suatu bagian yang nantinya akan menentukan pekerjaan – pekerjaan apa saja yang dapat dikerjakan oleh bagian ke dua, yaitu Worker. Model ini dapat dianalogikan sebagai hubungan antara pekerja dan seorang manager pada suatu kantor pada umumnya.

Pada model Boss – Worker, bagian pertama yaitu boss hanya memiliki pekerjaan yang relatif sedikit. Thread boss, atau thread utama hanya menerima request yang masuk dari aplikasi client yang kemudian akan didistribusikan pada thread – thread pekerja yang dibuat oleh thread boss. Berikut merupakan bagan sederhana yang menggambarkan keseluruhan model boss – worker :



Gambar 2.6 Bagan model *Boss worker*

Thread worker akan menerima request pekerjaan yang telah didistribusikan oleh thread Boss sehingga tidak perlu lagi menentukan pekerjaan yang diambil untuk diproses. Pada model boss – worker ini terdapat banyak thread worker yang bergantung pada thread boss. Jarang ada thread worker yang bergerak sendiri tanpa perintah dari thread boss / thread utama. Berikut merupakan gambar yang menjelaskan prinsip kerja boss worker thread secara keseluruhan.



Gambar 2.7 Diagram prinsip kerja model *boss worker*

Pada gambar tersebut, data input yang diterima oleh Boss Thread akan didistribusikan kepada thread pekerja yang sudah ada berdasarkan aturan atau algoritma yang telah ditentukan pada aplikasi tersebut. Setelah

worker thread selesai melakukan pemrosesan maka worker thread akan mengirim outputnya (hasil pemrosesan) kepada Boss Thread untuk pemrosesan data yang lebih lanjut apabila ada.

Model boss – worker merupakan suatu model yang sering digunakan untuk suatu aplikasi yang berbasis GUI (Graphic User Interface). Hal ini dikarenakan bukan karena kecepatan pengerjaan tiap pemrosesan tetapi karena model boss – worker memiliki kecenderungan untuk memiliki waktu response yang bagus antara user dan aplikasi.

2.10.2. Kegunaan dan keuntungan *Multithreading*

Multithreading merupakan model programming dan pengeksekusian yang populer yang memungkinkan untuk banyak thread pada suatu process. Thread tersebut merupakan bagian dari process yang sama tapi dapat dieksekusikan secara terpisah. Penggunaan teknologi ini dapat digunakan untuk membuat sebuah process dieksekusikan pada banyak processor.

Keuntungan tersebut menjadikan pengeksekusian suatu operasi berjalan lebih cepat pada komputer yang memiliki banyak CPU, CPU yang memiliki banyak inti (core), atau operasi antar banyak mesin(cluster of machines). Hal ini dikarenakan thread dalam program dijalankan secara paralel.

Keuntungan lain yang diperoleh dengan teknologi multithreading, walaupun hanya dengan suatu komputer yang memiliki sebuah CPU, yaitu

kemampuan untuk sebuah aplikasi untuk tetap responsif terhadap input. Pada sebuah program yang memiliki sebuah thread , jika sedang mengeksekusikansebuah taks yang lama, seluruh aplikasi tersebut akan tampak berhenti (*freeze*). Dengan memindahkan task tersebut ke worker thread yang berjalan secara paralel dengan eksekusi thread utamanya, sangat memungkinkan untuk aplikasi tersebut untuk tetap responsif terhadap input dari user selama mengerjakan task tersebut di latar (background).

Pada sebuah sistem operasi penjadwalan thread ada dua cara. *Preemptive multithreading* yang pada dianggap pendekatan yang lebih unggul, hal ini dikarenakan sistem operasi dapat menentukan kapan context switch harus dilakukan. *Cooperative multithreading*, merupakan suatu pendekatan dimana thread tersebut yang menentukan kapan thread tersebut akan diberhentikan.

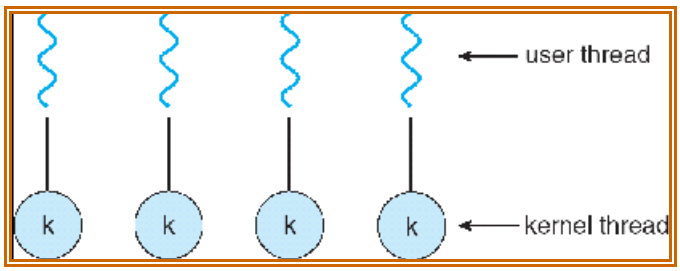
Perangkat keras komputer tradisional tidak memiliki banyak dukungan untuk fasilitas multithreading hal ini dikarenakan perpindahan antar satu thread dengan thread yang lain dianggap lebih cepat dibandingkan keseluruhan proses context switch. Pada akhir 1990, suatu ide untuk menjalankan instruksi menggunakan banyak thread secara bersamaan dikenal dengan nama simultaneous multithreading. Fitur ini diperkenalkan oleh perusahaan Intel dengan nama teknologi hyper threading.

2.10.3 Berbagai Model Thread

Terdapat berbagai model thread yang diguanakan, antara lain :

1:1

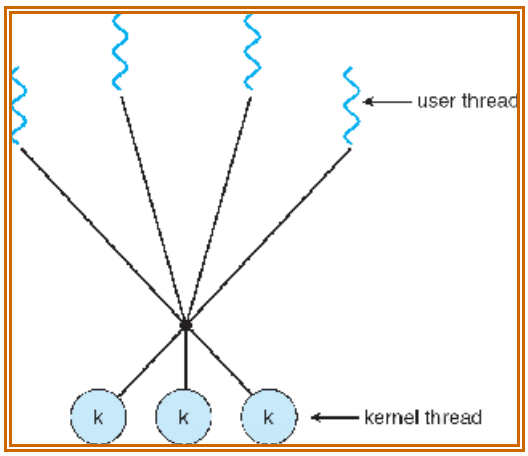
Thread yang dibuat oleh user merupakan korespondensi 1 – 1 dengan entitas yang dapat dijadwalkan pada kernel.



Gambar 2.8 Model *thread* satu ke satu

N:M

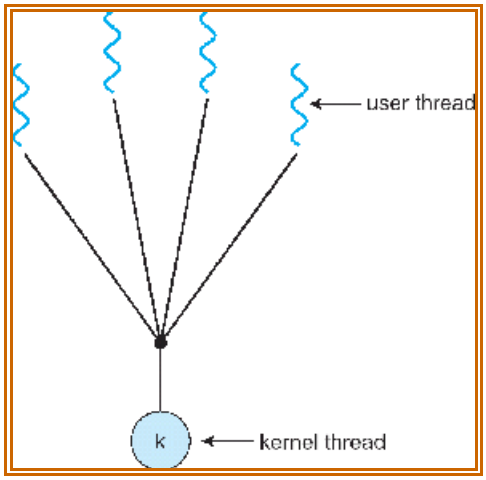
Memetakan N banyak thread aplikasi untuk dieksekusikan oleh M banyak nya entitas kernel, atau “virtual processor”.



Gambar 2.9 Model *thread* banyak ke banyak

N:1

Semua thread dengan level aplikasi dapat dipetakan atau dijadwalkan pada sebuah kernel.

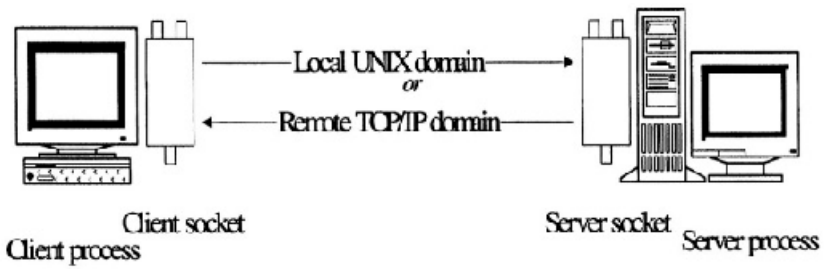


Gambar 2.10 Model *thread* satu ke banyak

2.11 *Sockets*

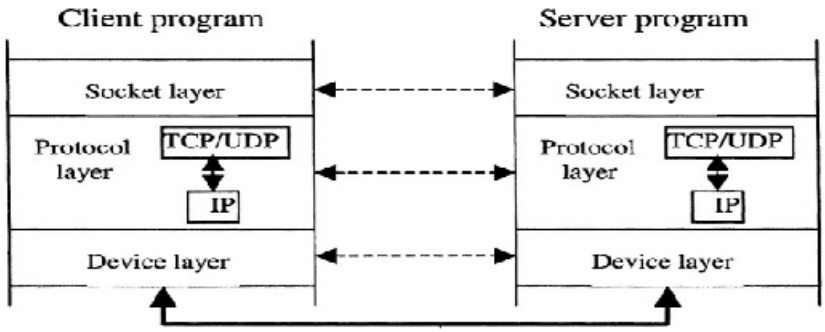
Pada software jaringan TCP/IP pada umumnya mendukung adanya banyak jenis antar muka untuk aplikasi tersebut untuk dapat berkomunikasi melalui internet.

Pada umumnya socket merupakan sebuah abstraksi dari I/O. Hal ini dapat digambarkan sebagai berikut :



Gambar 2.11 Hubungan antar *socket* pada *server client*

Sebuah server pertama akan menciptakan sebuah socket yang akan mendengarkan dan menunggu adanya request dari client. Pada saat client tersebut menciptakan sebuah socket yang akan mengirim permintaan koneksi pada server, server tersebut akan menciptakan suatu koneksi antara server dan client.



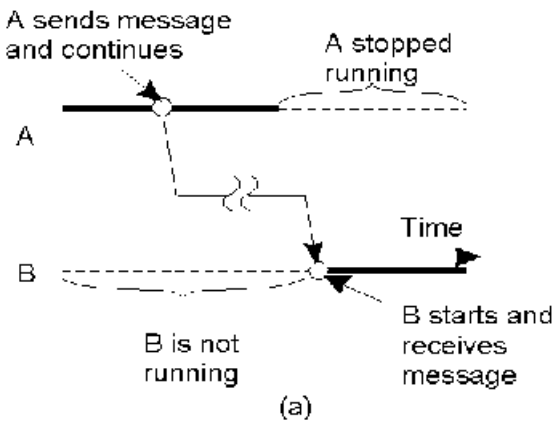
Gambar 2.12 Hubungan *socket* dengan layer protokol

2.12. Komunikasi Data

Dalam sistem terdistribusi terdapat beragam konsep komunikasi data. Berikut merupakan komunikasi data yang dapat terjadi antara satu mesin dengan mesin yang lainnya

2.12.1 *Persistent asynchronous communication*

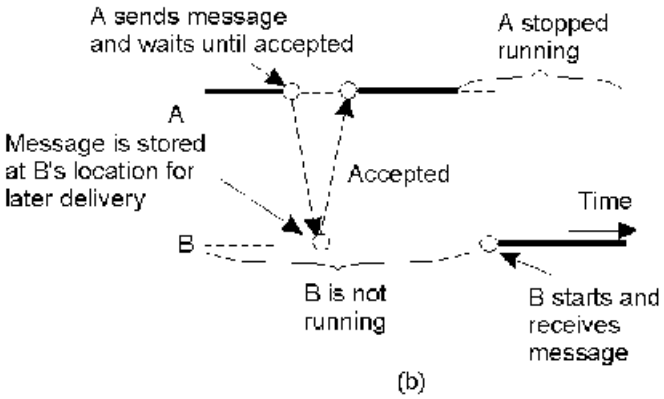
Pada saat A mengirim pesan, proses pada A tetap berjalan hingga B menerima pesan. Pada saat B sudah menerima pesan proses di A dihentikan dan proses di B mulai dijalankan.



Gambar 2.13 ***Persistent asynchronous communication***

2.12.2 *Persistence synchronous communication*

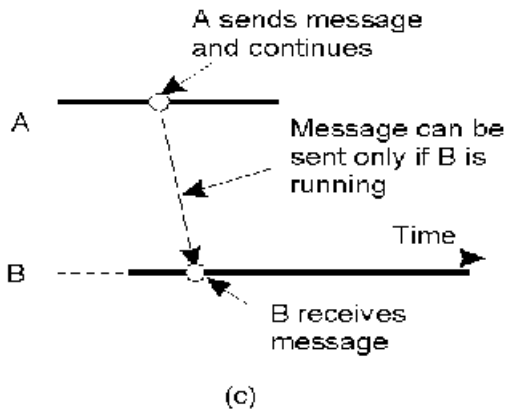
Pada saat A mengirim pesan, proses di A dihentikan dan menunggu respons dari B. Ketika B menerima pesan dari A, proses di B tidak langsung dijalankan, tetapi menunggu konfirmasi dari A. Ketika proses di A dihentikan maka B baru akan berjalan.



Gambar 2.14 ***Persistence synchronous communication***

2.12.3 *Transient asynchronous communication*

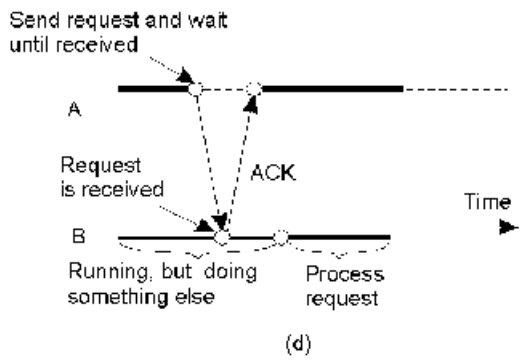
Pada saat A mengirim pesan ke B, B menerima pesan tetapi hanya jika B sedang berjalan.



Gambar 2.15 ***Transient asynchronous communication***

2.12.4. *Receipt – based transient synchronous communication*

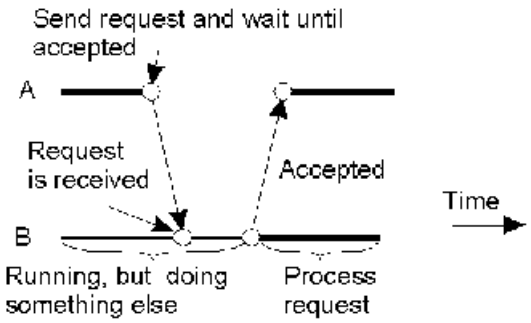
Pada saat A mengirim pesan ke B, A menunggu. B yang pada saat itu sedang mengerjakan proses lain akan menerima. Pada saat pesan sudah diterima B, a akan meneruskan kembali proses yang sedang berjalan.



Gambar 2.16 ***Receipt – based transient synchronous communication***

2.12.5. *Delivery-based transient synchronous communication*

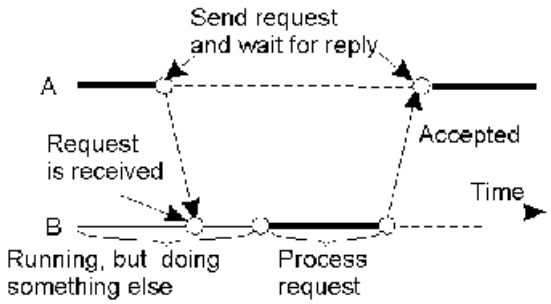
Saat A mengirim pesan, B yang pada saat itu sedang berjalan akan menerima request. Bakan mengirim pesan ke A yang menyatakan pesan diterima dan langsung mengerjakan *request*. Pada saat konfirmasi dari B diterima oleh A makan A akan melanjutkan proses.



Gambar 2.17 ***Delivery-based transient synchronous communication***

2.12.6. *Response-based transient synchronous communication*

Pada saat A mengirim pesan kepada B yang pada saat itu sedang mengerjakan pekerjaan lain, A menunggu konfirmasi dari B setelah pesan dari A sudah selesai dikerjakan. A akan menunggu selama proses itu dikerjakan oleh B.



Gambar 2.18 ***Response-based transient synchronous communication***

2.13 Teknik Pengujian Perangkat Lunak

Pengujian perangkat lunak adalah elemen kritis dari jaminan kualitas perangkat lunak dan mempesentasikan kajian pokok dari spesifikasi, desain, dan pengkodean.

2.13.1 Dasar Pengujian Perangkat Lunak

Pengujian menyajilkan anomali yang menarik bagi perekayasa perangkat lunak. Pada proses perangkat lunak, perekayasa pertama-tama berusaha membangun perangkat lunak dari konsep abstrak ke implementasi yang dapat dilihat, baru dilakukan pengujian. Perekayasa menciptakan sederetan *test case* yang dimaksudkan untuk "membongkar" perangkat lunak yang sudah dibangun. Pada dasarnya pengujian merupakan salah satu langkah dlam proses rekayasa perangkat lunak yang dianggap sebagai hal yang destruktif daripada konstruktif.

2.13.2 Sasaran-sasaran Pengujian

Dalam buku klasiknya mengenai pengujian perangkat lunak, Glen Myers menyatakan sejumlah aturan yang berfungsi sebagai sasaran pengujian:

1. Pengujian adalah proses eksekusi suatu program dengan meksud menemukan kesalahan.
2. *Test case* yang baik adalah *test case* yang memiliki probabilitas tinggi untuk menemukan kesalahan yang belum ditemukan sebelumnya.
3. Pengujian yang sukses adalah pengujian yang mengungkap semua kesalahan yang belum pernah ditemukan sebelumnya.

2.13.3 Prinsip Pengujian

Sebelum mengaplikasikan metode untuk mendesain *test case* yang efektif, perekayasa harus memahami prinsip dasar yang menuntun pengujian perangkat lunak. Davis mengusulkan serangkaian prinsip-prinsip pengujian diantaranya:

1. Semua pengujian harus dapat ditelusuri sampai kepesyaraan pelanggan.
2. Pengujian harus direncanakan lama sebelum pengujian itu mulai.
3. Prinsip pareto berlaku untuk pengujian perangkat lunak.
4. Pengujian harus mulai dari yang kecil dan yang berkembang kepengujian yanglebih besar.
5. Pengujian yang mendalam tidak mungkin.

6. Untuk menjadi paling efektif pengujian harus dilakukan oleh pihak ketiga yang *independen.*

2.13.4 Testabilitas

Testibilitas perangkat lunak adalah seberapa mudah program komputer dapat diuji. Karena pengujian sulitk, maka perlu diketahui apa yang harus dilakukan agar manjadi lebih mudah. *Cheklist* berikut ini memberikan serangkaian karakteristik yag membawa peragkat lunak yang dapat diuji.

1. *Operabilitas,* “semakin baik dia bekerja, semakin efisien dia diuji”.
2. *Observabilitas,* “apa yanganda lihat adalah apa yang anda uji”.
3. *Kontrabilitas,* semakin baik kita dapat mengontrol perangkat lunak semakin banyak pengujian yang diotomatisasi dan dioptimalkan”.
4. *Dekomposabilitas,* “dengan mengontrol ruang lingkup pengujian, kita dapat lebih cepat mengisolasi masalah dan melakukan pengujian kembali secara lebih halus”.
5. Kesederhanaan, “semakin cepat yang diuji, semakin sedikit kita dapat mengujinya”.
6. *Stabilitas,* “semakin sedikit perubahan, semakin gagguan dalam pengujian”.
7. *Verifikasi,* mengacu kepada rangkaian aktivitas yang memastikan bahwa perangkat lunak secara tepatmengimplementasikan suatu fungsi terentu.

8. *Validasi*, mengacu pada rangkaian aktivitas berbeda yang memastikan bahwa prangkat lunak yang dibangun dapat ditelusuri kepersyaratan pelanggan. “Apakah kita membangun produk yang benar”.

2.13.5 Pengujian *Black Box*

Pengujian *Black-box* berfokus pada persyaratan fungsional perangkat lunak. Dengan demikian, pengujian *black-box* memungkinkan perekayasa peangkat lunak mendapatkan serangkaian kondisi input yang sepenuhnya semua persyaratan fungsional untuk suatu program.

Penguian *black-box* berusaha menemukan kesalahan dalam kategori sebagai berikut:

1. Fungsi-fungsi yang tidak benar atau hilang.
2. Kesalahan *Interface.*
3. Kesalahan dalam struktur data atau akses *database* eksternal.
4. Kesalahan kinerja.
5. Inisialisasi dan kesalahan terminasi.

This document was created with Win2PDF available at http://www.daneprairie.com.
The unregistered version of Win2PDF is for evaluation or non-commercial use only.