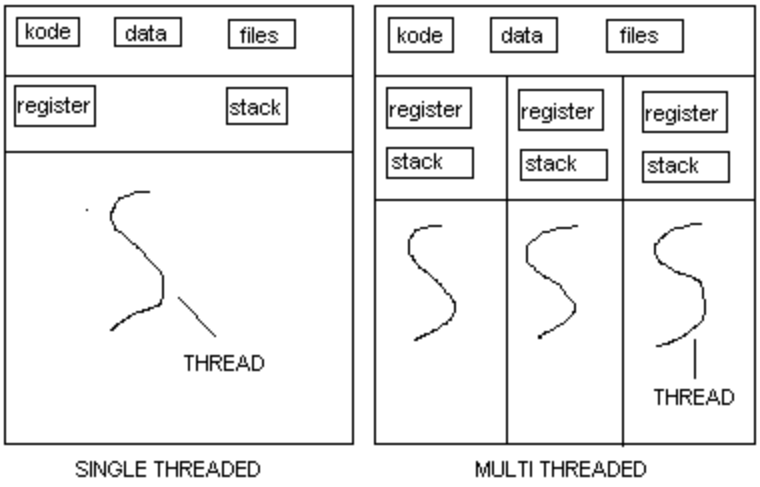
Bab 3. Proses

3.1 Thread

Thread adalah sebuah alur kontrol dari sebuah proses. Kontrol *thread* tunggal ini hanya memungkinkan proses untuk menjalankan satu tugas pada satu waktu. Banyak sistem operasi modern telah memiliki konsep yang dikembangkan agar memungkinkan sebuah proses untuk memiliki eksekusi *multi-threads*, agar dapat secara terus menerus mengetik dan menjalankan pemeriksaan ejaan didalam proses yang sama, maka sistem operasi tersebut memungkinkan proses untuk menjalankan lebih dari satu tugas pada satu waktu. Suatu proses yang *multithreaded* mengandung beberapa perbedaan alur kontrol dengan ruang alamat yang sama.

Keuntungan dari *multithreaded* meliputi peningkatan respon dari pengguna, pembagian sumber daya proses, ekonomis, dan kemampuan untuk mengambil keuntungan dari arsitektur multiprosesor. *Thread* merupakan unit dasar dari penggunaan CPU, yang terdiri dari Thread_ID, *program counter*, *register set*, dan *stack*. Sebuah *thread* berbagi *code section*, *data section*, dan sumber daya sistem operasi dengan Thread lain yang dimiliki oleh proses yang sama. Thread juga sering disebut *lightweight process*. Sebuah proses tradisional atau *heavyweight process* mempunyai *thread* tunggal yang berfungsi sebagai pengendali.

Perbedaan antara proses dengan *thread* tunggal dan proses dengan *thread* yang banyak adalah proses dengan *thread* banyak dapat mengerjakan lebih dari satu tugas pada satu satuan waktu.



Gambar 3.1. Thread

Banyak perangkat lunak yang berjalan pada PC modern dirancang secara *multi-threading*. Sebuah aplikasi biasanya diimplementasi sebagai proses yang terpisah dengan beberapa *thread* yang berfungsi sebagai pengendali. Contohnya sebuah *web browser* mempunyai *thread* untuk menampilkan gambar atau tulisan sedangkan *thread* yang lain berfungsi sebagai penerima data dari network.

Kadang kala ada situasi dimana sebuah aplikasi diperlukan untuk menjalankan beberapa tugas yang serupa. Sebagai contohnya sebuah *web server* dapat mempunyai ratusan klien yang mengaksesnya secara *concurrent*. Kalau *web server* berjalan sebagai proses yang hanya mempunyai *thread* tunggal maka ia hanya dapat melayani

satu klien pada pada satu satuan waktu. Bila ada klien lain yang ingin mengajukan permintaan maka ia harus menunggu sampai klien sebelumnya selesai dilayani. Solusinya adalah dengan membuat *web server* menjadi *multi-threading*. Dengan ini maka sebuah *web server* akan membuat *thread* yang akan mendengar permintaan klien, ketika permintaan lain diajukan maka *web server* akan menciptakan *thread* lain yang akan melayani permintaan tersebut.

Keuntungan *Thread*

Keuntungan dari program yang *multithreading* dapat dipisah menjadi empat kategori:

1. **Responsi**: Membuat aplikasi yang interaktif menjadi *multithreading* dapat membuat sebuah program terus berjalan meski pun sebagian dari program tersebut diblok atau melakukan operasi yang panjang, karena itu dapat meningkatkan respons kepada pengguna. Sebagai contohnya dalam *web browser* yang *multithreading*, sebuah *thread* dapat melayani permintaan pengguna sementara *thread* lain berusaha menampilkan image.

2. **Berbagi sumber daya**: *thread* berbagi memori dan sumber daya dengan *thread* lain yang dimiliki oleh proses yang sama. Keuntungan dari berbagi kode adalah mengizinkan sebuah aplikasi untuk mempunyai beberapa *thread* yang berbeda dalam lokasi memori yang sama.

3. **Ekonomi**: dalam pembuatan sebuah proses banyak dibutuhkan pengalokasian memori dan sumber daya. Alternatifnya adalah dengan penggunaan *thread*, karena *thread* berbagi memori dan sumber daya proses yang memilikinya maka akan lebih ekonomis untuk membuat dan *context switch thread*. Akan susah untuk mengukur perbedaan waktu antara proses dan *thread* dalam hal pembuatan dan pengaturan, tetapi secara umum pembuatan dan pengaturan proses lebih lama dibandingkan *thread*. Pada Solaris, pembuatan proses lebih lama 30 kali dibandingkan pembuatan *thread*, dan *context switch* proses 5 kali lebih lama dibandingkan *context switch thread*.

4. **Utilisasi arsitektur *multiprocessor***: Keuntungan dari multithreading dapat sangat meningkat pada arsitektur *multiprocessor*, dimana setiap *thread* dapat berjalan secara pararel di atas processor yang berbeda. Pada arsitektur processor tunggal, CPU menjalankan setiap *thread* secara bergantian tetapi hal ini berlangsung sangat cepat sehingga menciptakan ilusi pararel, tetapi pada kenyataannya hanya satu *thread* yang dijalankan CPU pada satu-satuan waktu (satu-satuan waktu pada CPU biasa disebut *time slice* atau *quantum*).

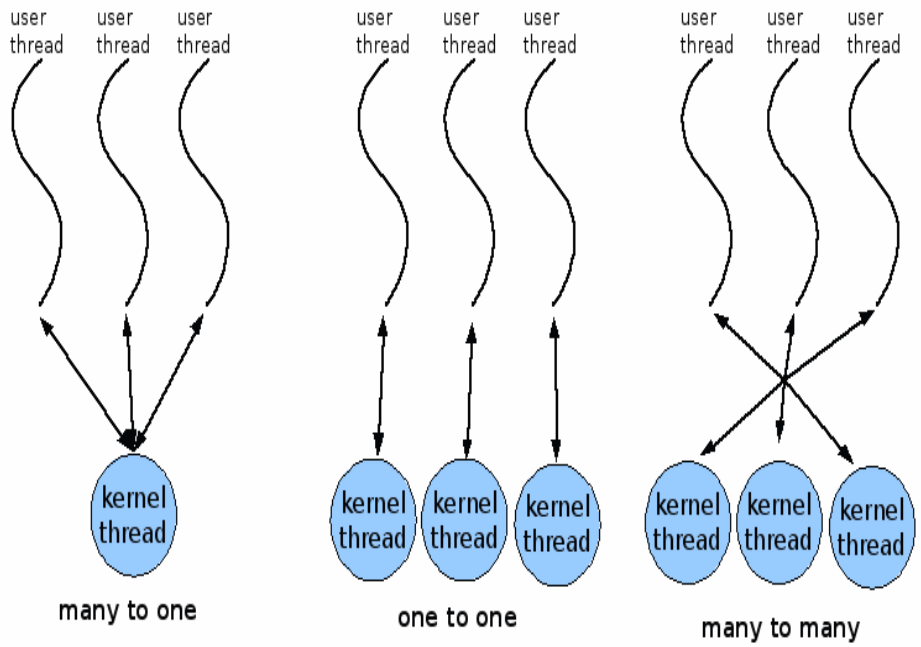
Multithreading Models

Beberapa terminologi yang akan dibahas:

a. ***Thread* pengguna:** *Thread* yang pengaturannya dilakukan oleh pustaka *thread* pada tingkatan pengguna. Karena pustaka yang menyediakan fasilitas untuk pembuatan dan penjadwalan *thread*, *thread* pengguna cepat dibuat dan dikendalikan.
b. ***Thread* Kernel: .** *Thread* yang didukung langsung oleh kernel. Pembuatan, penjadwalan dan manajemen *thread* dilakukan oleh kernel pada *kernel space*. Karena dilakukan oleh sistem operasi, proses pembuatannya akan lebih lambat jika dibandingkan dengan *thread* pengguna.

Model-model *Multithreading*:

a. **Model *Many-to-One*.** Model ini memetakan beberapa *thread* tingkatan pengguna ke sebuah *thread*. tingkatan kernel. Pengaturan *thread* dilakukan dalam ruang pengguna sehingga efisien. Hanya satu *thread* pengguna yang dapat mengakses *thread* kernel pada satu saat. Jadi *Multiple thread* tidak dapat berjalan secara paralel pada multiprosesor. Contoh: Solaris *Green Threads* dan GNU *Portable Threads*.
b. **Model *One-to-One*.** Model ini memetakan setiap *thread* tingkatan pengguna ke setiap *thread*. Ia menyediakan lebih banyak *concurrency* dibandingkan model *Many-to-One*. Keuntungannya sama dengan keuntungan *thread* kernel. Kelemahan model ini ialah setiap pembuatan *thread* pengguna memerlukan tambahan *thread* kernel. Karena itu, jika mengimplementasikan sistem ini maka akan menurunkan kinerja dari sebuah aplikasi sehingga biasanya jumlah *thread* dibatasi dalam sistem. Contoh: Windows NT/XP/2000 , Linux, Solaris 9.
c. **Model *Many-to-Many*.** Model ini memultipleks banyak *thread* tingkatan pengguna ke *thread* kernel yang jumlahnya sedikit atau sama dengan tingkatan pengguna. Model ini mengizinkan *developer* membuat *thread* sebanyak yang ia mau tetapi *concurrency* tidak dapat diperoleh karena hanya satu *thread* yang dapat dijadwalkan oleh kernel pada suatu waktu. Keuntungan dari sistem ini ialah kernel *thread* yang bersangkutan dapat berjalan secara paralel pada multiprosessor.



Pustaka *Thread*

Pustaka *Thread* atau yang lebih familiar dikenal dengan *Thread Library* bertugas untuk menyediakan API untuk *programmer* dalam menciptakan dan memanage *thread*. Ada dua cara dalam mengimplementasikan pustaka *thread*:

a. Menyediakan API dalam level pengguna tanpa dukungan dari kernel sehingga pemanggilan fungsi tidak melalui *system call*. Jadi, jika kita memanggil fungsi yang sudah ada di pustaka, maka akan menghasilkan pemanggilan fungsi *call* yang sifatnya lokal dan bukan *system call*.
b. Menyediakan API di level kernel yang didukung secara langsung oleh sistem operasi. Pemanggilan fungsi *call* akan melibatkan *system call* ke kernel.

Ada tiga pustaka *thread* yang sering digunakan saat ini, yaitu: POSIX Pthreads, Java, dan Win32. Implementasi POSIX standard dapat dengan cara *user* level dan kernel level, sedangkan Win32 adalah kernel level. Java API *thread* dapat diimplementasikan oleh Pthreads atau Win32.

Pembatalan *Thread* (*Thread Cancellation*)

Thread Cancellation ialah pembatalan *thread* sebelum tugasnya selesai. Umpamanya, jika dalam program Java hendak mematikan Java Virtual Machine (JVM). Sebelum JVM dimatikan, maka seluruh *thread* yang berjalan harus dibatalkan terlebih dahulu. Contoh lain adalah di masalah *search*. Apabila sebuah *thread* mencari sesuatu dalam *database* dan menemukan serta mengembalikan hasilnya, *thread* sisanya akan dibatalkan. *Thread* yang akan diberhentikan biasa disebut *target thread*.

Pemberhentian *target Thread* dapat dilakukan dengan 2 cara:

a. ***Asynchronous cancellation*.** Suatu *thread* seketika itu juga membatalkan *target thread*.
b. ***Deferred cancellation*.** Suatu *thread* secara periodik memeriksa apakah ia harus batal, cara ini memperbolehkan *target thread* untuk membatalkan dirinya secara terurut.

Hal yang sulit dari pembatalan *thread* ini adalah ketika terjadi situasi dimana sumber daya sudah dialokasikan untuk *thread* yang akan dibatalkan. Selain itu kesulitan lain adalah ketika *thread* yang dibatalkan sedang meng-*update* data yang ia bagi dengan *thread* lain. Hal ini akan menjadi masalah yang sulit apabila digunakan *asynchronous cancellation*. Sistem operasi akan mengambil kembali sumber daya dari *thread* yang dibatalkan tetapi seringkali sistem operasi tidak mengambil kembali semua sumber daya dari *thread* yang dibatalkan.

Alternatifnya adalah dengan menggunakan *deffered cancellation*. Cara kerja dari *deffered cancellation* adalah dengan menggunakan satu *thread* yang berfungsi sebagai pengindikasi bahwa *target thread* hendak dibatalkan. Tetapi pembatalan hanya akan terjadi jika *target thread* memeriksa apakah ia harus batal atau tidak. Hal ini memperbolehkan *thread* untuk memeriksa apakah ia harus batal pada waktu dimana ia dapat dibatalkan secara aman yang aman. Pthread merujuk sebagai *cancellation points*.

Pada umumnya sistem operasi memperbolehkan proses atau thread untuk dibatalkan secara *asynchronous*. Tetapi Pthread API menyediakan *deferred cancellation*. Hal ini berarti sistem operasi yang mengimplementasikan Pthread API akan mengizinkan *deferred cancellation*.

Penjadwalan *Thread*

Begitu dibuat, *thread* baru dapat dijalankan dengan berbagai macam penjadwalan. Kebijakan penjadwalanlah yang menentukan setiap proses, di mana proses tersebut akan ditaruh dalam daftar proses sesuai proritasnya dan bagaimana ia bergerak dalam daftar proses tersebut.

Untuk menjadwalkan *thread*, sistem dengan model *mulithreading many to many* atau *many to one* menggunakan:

a. **Process Contention Scope (PCS).** Pustaka *thread* menjadwalkan thread pengguna untuk berjalan pada LWP (*lightweight process*) yang tersedia.
b. **System Contention Scope (SCS).** SCS berfungsi untuk memilih satu dari banyak *thread* , kemudian menjadwalkannya ke satu *thread* tertentu(CPU / Kernel).

3.2. Client – Server

Pengertian Client Server

Banyak referensi yang menjelaskan istilah client server. Menurut definisi yang diperoleh dari Wikipedia:

> ***Client server*** *is network architecture which separates a client (often an application that uses a graphical user interface) from a server. Each instance of the client software can send requests to a server. Specific types of servers include web servers, application servers, file servers, terminal servers, and mail servers. While their purposes vary somewhat, the basic architecture remains the same.*

Menurut Microsoft Encarta:

> *Definition:* ***servicing requests from others:*** *describes a computer network in which processing is divided between a client program running on a user's machine and a network server program. One server can provide data to, or perform storage-intensive processing tasks in conjunction with, one or more clients.*

Dari buku **Technical Foundations of Client/Server Systems** karangan Carl L. Hall (A Wiley – QED Publication). Di halaman 7 tentang Client/Server General Definition disebutkan:

> *The term client/server refers to a relationship between two systems or processes. The client is the system that requests work to be done on its behalf by the server system. In most situations, which is client and which is server is determined by the relationship of requester (client) to server.*

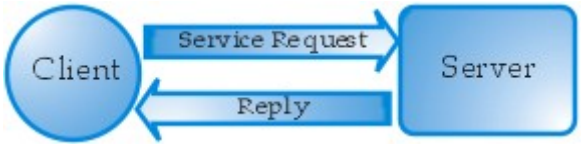
Dari definisi-definisi tersebut, kata kuncinya adalah pada sistem client/server harus terdapat **satu atau beberapa server yang menyediakan layanan** dan **satu atau beberapa klien yang meminta layanan** tersebut (tidak perduli apakah kondisi tersebut berada pada sebuah sistem jaringan ataupun stand-alone).

Istilah server di sini bisa saja berupa komputer-komputer kelas server seperti IBM, HP, Compaq dll. Atau juga berupa software yang dapat dikategorikan

berdasarkan layanannya misalnya web server, application server, file server, database server, terminal server, mail server, dll.

Server bisa juga berupa proses, seperti RPC Server yang terdapat pada sistem operasi server seperti Novell, Windows NT, Linux dll. Lebih dalam lagi, pada kernel (inti) sebuah sistem operasi juga banyak terdapat proses-proses yang bertanggung-jawab menyediakan layanan-layanan agar hardware komputer dapat bekerja sebagai mana mestiya.

Microsoft menamakan proses tersebut *services* sedangkan keluarga Unix/Linux menyebutnya *daemons.* Services/daemons tersebut umumnya menyediakan manajemen memory, akses file/jaringan, serta penjadwalan (scheduling).



Server adalah komputer yang dapat memberikan *service* ke *client,* sedangkan *client* adalah komputer yang mengakses beberapa *service* yang ada di *server*. Ketika *client* membutuhkan suatu *service* yang ada di *server*, dia akan mengirim *request* kepada *server* lewat jaringan. *Jika request* tersebut dapat dilaksanakan, maka *server* akan mengirim balasan berupa *service* yang dibutuhkan untuk saling berhubungan menggunakan *Socket.*

1. Karakteristik *Server*
    a. Pasif
    b. Menunggu *request*
    c. Menerima *request*, memproses mereka dan mengirimkan balasan berupa *service*
2. Karakteristik *Client*
    a. Aktif
    b. Mengirim *request*
    c. Menunggu dan menerima balasan dari *server*

Socket adalah sebuah *endpoint* untuk komunikasi didalam jaringan. Sepasang proses atau *thread* berkomunikasi dengan membangun sepasang *socket*, yang masing-masing proses memilikinya. *Socket* dibuat dengan menyambungkan dua buah alamat IP melalui *port* tertentu. Secara umum *socket* digunakan dalam *client/server system*, dimana sebuah *server* akan menunggu *client* pada *port* tertentu. Begitu ada *client* yang menghubungi *server* maka *server* akan menyetujui komunikasi dengan *client* melalui *socket* yang dibangun.

Model Client-Server

Ada beberapa model client/server yang penting untuk diketahui. Dimulai dari arsitektur mainframe hingga arsitektur client/server.

a. **Arsitektur Mainframe**

Pada arsitektur ini, terdapat sebuah komputer pusat (host) yang memiliki sumber daya yang sangat besar, baik memori, processor maupun media penyimpanan. Melalui komputer terminal, pengguna mengakses sumber daya tersebut. Komputer terminal hanya memiliki monitor/keyboard dan tidak memiliki CPU. Semua sumber daya yang diperlukan terminal dilayani oleh komputer host. Model ini berkembang pada akhir tahun 1980-an.

b. **Arsitektur File Sharing**

Pada arsitektur ini komputer server menyediakan file-file yang tersimpan di media penyimpanan server yang dapat diakses oleh pengguna. Arsitektur file sharing memiliki keterbatasan, terutama jika jumlah pengakses semakin banyak serta ukuran file yang di shaing sangat besar. Hal ini dapat mengakibatkan transfer data menjadi lambat. Model ini populer pada tahun 1990-an.

c. **Arsitektur Client/Server**

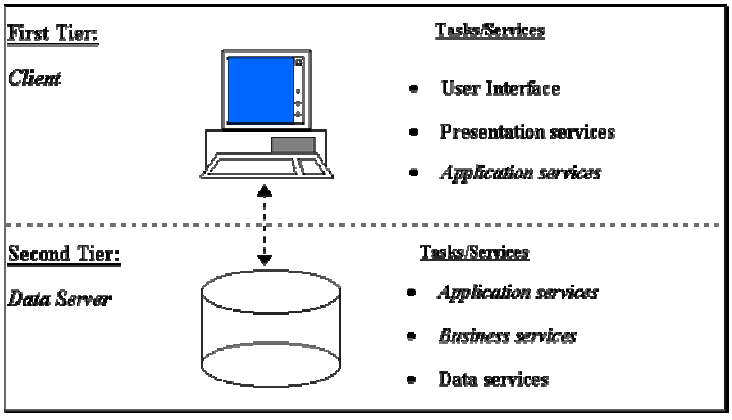
Karena keterbatasan sistem file sharing, dikembangkanlah arsitektur client/server. Salah satu hasilnya yaitu berupa software database server yang menggantikan software database berbasis file server. Dikenalkan pula RDBMS (Relational Database Management System). Dengan arsitektur ini, query data ke server dapat terlayani dengan lebih cepat karena yang ditransfer bukanlah file, tetapi hanyalah hasil dari query tersebut. RPC (Remote Procedure Calls) memegang peranan penting pada arsitektur client/server.

d. **Model Two-tier**

Model Two-tier terdiri dari tiga komponen yang disusun menjadi dua lapisan: *client* (yang meminta *service*) dan *server* (yang menyediakan *service*). Tiga komponen tersebut yaitu :

1. User Interface, yaitu antar muka program aplikasi yang berhadapan dan digunakan langsung oleh user.
2. Manajemen proses
3. Database

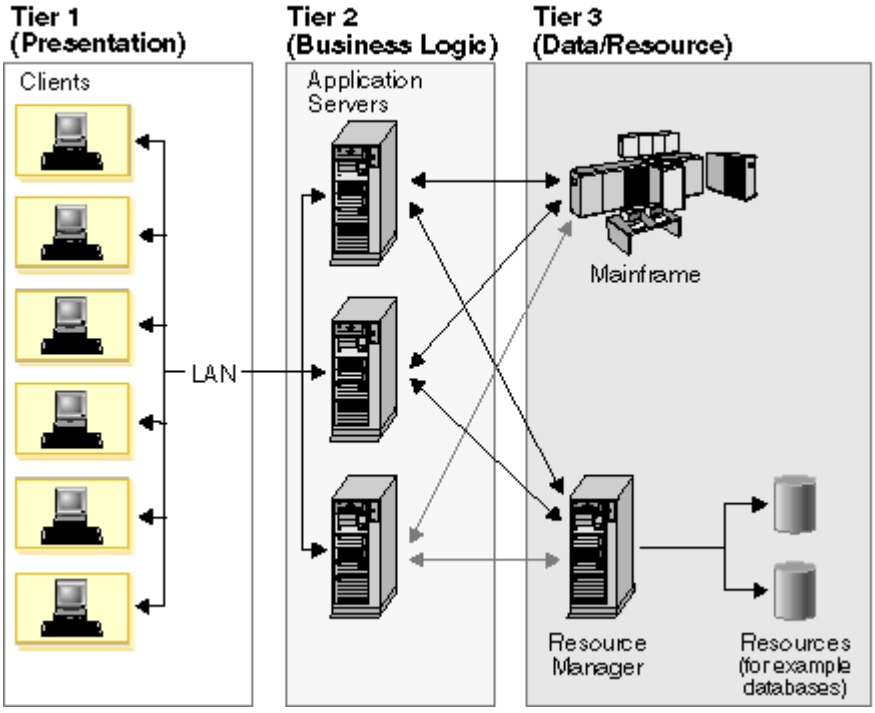
Model ini memisahkan peranan user interface dan database dengan jelas, sehingga terbentuk dua lapisan.



Pada gambar tersebut, user interface yang merupakan bagian dari program aplikasi melayani input dari user. Input tersebut diproses oleh Manajemen Proses dan melakukan query data ke database (dalam bentuk perintah SQL). Pada database server juga bisa memiliki Manajemen Proses untuk melayani query tersebut, biasanya ditulis ke dalam bentuk Stored Procedure.

e. **Model Three-tier**

Pada model ini disisipkan satu layer tambahan diantara user interface tier dan database tier. Tier tersebut dinamakan middle-tier. Middle-Tier terdiri dari bussiness logic dan rules yang menjembatani query user dan database, sehingga program aplikasi tidak bisa mengquery langsung ke database server, tetapi harus memanggil prosedur-prosedur yang telah dibuat dan disimpan pada middle-tier. Dengan adanya server middle-tier ini, beban database server berkurang. Jika query semakin banyak dan/atau jumlah pengguna bertambah, maka server-server ini dapat ditambah, tanpa merubah struktur yang sudah ada. Ada berbagai macam software yang dapat digunakan sebagai server middle-tier. Contohnya MTS (Microsoft Transaction Server) dan MIDAS.



3.3. Agent

Software Agent adalah entitas perangkat lunak yang didedikasikan untuk tujuan tertentu yang memungkinkan user untuk mendelegasikan tugasnya secara mandiri, selanjutnya software agent nantinya disebut agent saja. Agent bisa memiliki ide sendiri mengenai bagaimana menyelesaikan suatu pekerjaan tertentu atau agenda tersendiri. Agent yang tidak berpindah ke host lain disebut *stationary agent.*

Karakteristik dari Agen:

1. ***Autonomy***: Agent dapat melakukan tugas secara mandiri dan tidak dipengaruhi secara langsung oleh user, agent lain ataupun oleh lingkungan

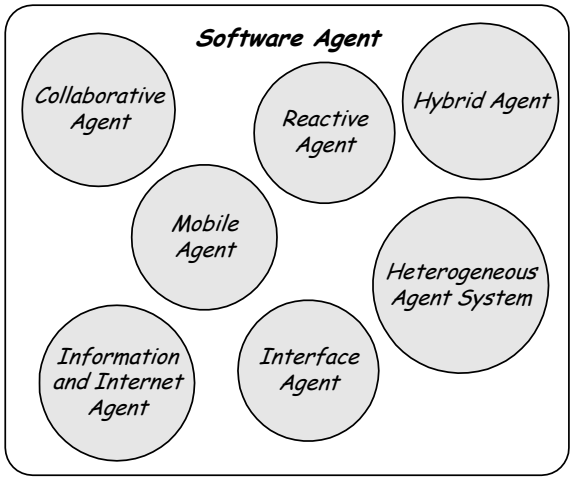
(environment). Untuk mencapai tujuan dalam melakukan tugasnya secara mandiri, agent harus memiliki kemampuan kontrol terhadap setiap aksi yang mereka perbuat, baik aksi keluar maupun ke dalam [Woolridge et. al., 1995].

2. ***Intelligence, Reasoning, dan Learning***: Setiap agent harus mempunyai standar minimum untuk bisa disebut agent, yaitu intelegensi (*intelligence*). Dalam konsep *intelligence*, ada tiga komponen yang harus dimiliki: internal *knowledge b*ase, kemampuan *reasoning* berdasar pada *knowledge base* yang dimiliki, dan kemampuan *learning* untuk beradaptasi dalam perubahan lingkungan.
3. ***Mobility dan Stationary***: Khusus untuk *mobile agent*, dia harus memiliki kemampuan yang merupakan karakteristik tertinggi yang dia miliki yaitu mobilitas. Berbeda dengan *stationary agent*. Tetapi keduanya tetap harus memiliki kemampuan untuk mengirim pesan dan berkomunikasi dengan agent lain.
4. ***Delegation***: Agent bergerak dalam kerangka menjalankan tugas yang diperintahkan oleh user. Fenomena pendelegasian (*delegation*) ini adalah **karakteristik utama** suatu program disebut agent.
5. ***Reactivity***: Kemampuan untuk bisa cepat beradaptasi dengan adanya perubahan informasi yang ada dalam suatu lingkungan. Lingkungan itu bisa mencakup: agent lain, user, informasi dari luar, dsb [Brenner et. al., 1998].
6. ***Proactivity dan Goal-Oriented***: Sifat *proactivity* boleh dibilang adalah kelanjutan dari sifat *reactivity*. Agent tidak hanya dituntut bisa beradaptasi terhadap perubahan lingkungan, tetapi juga harus mengambil inisiatif langkah penyelesaian apa yang harus diambil [Brenner et. al., 1998]. Untuk itu agent harus didesain memiliki tujuan (*goal*) yang jelas, dan selalu berorientasi kepada tujuan yang diembannya (*goal-oriented*).
7. ***Communication and Coordination Capability***: Agent harus memiliki kemampuan berkomunikasi dengan user dan juga agent lain. Masalah komunikasi dengan user adalah masuk ke masalah user interface dan perangkatnya, sedangkan masalah komunikasi, koordinasi, dan kolaborasi dengan *agent* lain adalah masalah sentral penelitian *Multi Agent System* (MAS). Bagaimanapun juga, untuk bisa berkoordinasi dengan *agent* lain dalam menjalankan tugas, perlu bahasa standard untuk berkomunikasi. Tim Finin [Finin et al., 1993] [Finin et al., 1994] [Finin et al., 1995] [Finin et al., 1997] dan Yannis Labrou [Labrou et al., 1994] [Labrou et al., 1997] adalah peneliti *software agent* yang banyak berkecimpung dalam riset mengenai bahasa dan protokol komunikasi antar agent. Salah satu produk mereka adalah *Knowledge Query and Manipulation Language* (KQML). Dan masih terkait dengan komunikasi antar agent adalah *Knowledge Interchange Format* (KIF).

Klasifikasi Software Agent

1. Klasifikasi menurut Karakteristik yang Dimiliki

Menurut Nwana, agent bisa diklasifikasikan menjadi delapan berdasarkan pada karakteristiknya.



Gambar 2: Klasifikasi *Software Agent* Menurut Karakteristik Yang Dimiliki

a. ***Collaborative Agent***: *Agent* yang memiliki kemampuan melakukan kolaborasi dan koordinasi antar agent dalam kerangka *Multi Agent System* (MAS).
b. ***Interface Agent***: *Agent* yang memiliki kemampuan untuk berkolaborasi dengan user, melakukan fungsi *monitoring* dan *learning* untuk memenuhi kebutuhan user.
c. ***Mobile Agent***: Agent yang memiliki kemampuan untuk bergerak dari suatu tempat ke tempat lain, dan secara mandiri melakukan tugas ditempat barunya tersebut, dalam lingkungan jaringan komputer.
d. ***Information dan Internet Agent***: *Agent* yang memiliki kemampuan untuk menjelajah internet untuk melakukan pencarian, pemfilteran, dan penyajian informasi untuk user, secara mandiri. Atau dengan kata lain, memanage informasi yang ada di dalam jaringan Internet.
e. ***Reactive Agent***: *Agent* yang memiliki kemampuan untuk bisa cepat beradaptasi dengan lingkungan baru dimana dia berada.
f. ***Hybrid Agent***: Kita sudah mempunyai lima klasifikasi *agent*. Kemudian *agent* yang memiliki katakteristik yang merupakan gabungan dari karakteristik yang sudah kita sebutkan sebelumnya adalah masuk ke dalam *hybrid agent*.
g. ***Heterogeneous Agent System***: Dalam lingkungan *Multi Agent System* (MAS), apabila terdapat dua atau lebih *hybrid agent* yang memiliki perbedaan kemampuan dan karakteristik, maka sistem MAS tersebut kita sebut dengan *heterogeneous agent system*.

2. Klasifikasi menurut Lingkungan Dimana Dijalankan

a. ***Desktop Agent***: *Agent* yang hidup dan bertugas dalam lingkungan *Personal Computer* (PC), dan berjalan diatas suatu *Operating System* (OS). Termasuk dalam klasifikasi ini adalah:
   - *Operating System Agent*
   - *Application Agent*
   - *Application Suite Agent*

b. ***Internet Agent***: *Agent* yang hidup dan bertugas dalam lingkungan jaringan Internet, melakukan tugas memanage informasi yang ada di Internet. Termasuk dalam klasifikasi ini adalah:
   - *Web Search Agent*
   - *Web Server Agent*
   - *Information Filtering Agent*
   - *Information Retrieval Agent*
   - *Notification Agent*
   - *Service Agent*
   - *Mobile Agent*

c. ***Intranet Agent***: *Agent* yang hidup dan bertugas dalam lingkungan jaringan Intranet, melakukan tugas memanage informasi yang ada di Intranet. Termasuk dalam klasifikasi ini adalah:
   - *Collaborative Customization Agent*
   - *Process Automation Agent*
   - *Database Agent*
   - *Resource Brokering Agent*

Bahasa Pemrograman yang digunakan

Bahasa pemrograman yang dipakai untuk tahap implementasi dari software agent, sangat menentukan keberhasilan dalam implementasi agent sesuai dengan yang diharapkan. Beberapa peneliti memberikan petunjuk tentang bagaimana karakteristik bahasa pemrorgaman yang sebaiknya di pakai [Knabe, 1995] [Brenner et al., 1998]. Diantaranya yaitu :

1. Object-Orientedness:

Karena agent adalah berhubungan dengan obyek, bahkan beberapa peneliti menganggap agent adalah obyek yang aktif, maka juga agent harus diimplementasikan kedalam pemrorgaman yang berorientasi obyek (object-oriented programming language).

2. Platform Independence:

Seperti sudah dibahas pada bagian sebelumnya, bahwa agent hidup dan berjalan diberbagai lingkungan. Sehingga idealnya bahasa pemrograman yang dipakai untuk implementasi adalah yang terlepas dari platform, atau dengan kata lain program tersebut harus bisa dijalankan di platform apapun (platform independence).

3. Communication Capability:

Pada saat berinteraksi dengan agent lain dalam suatu lingkungan jaringan (network environment), diperlukan kemampuan untuk melakukan komunikasi secara fisik. Sehingga diperlukan bahasa pemrograman yang dapat mensupport pemrograman yang berbasis network dan komunikasi.

4. Security:

Faktor keamanan (security) adalah factor yang sangat penting dalam memilih bahasa pemrorgaman untuk implementasi software agent. Terutama untuk mobil agent, diperlukan bahasa pemrograman yang mensupport level-level keamanan yang bisa membuat agent bergerak dengan aman.

5. Code Manipulation:

Beberapa aplikasi software agent memerlukan manipulasi kode program secara runtime, sehingga diperlikan bahasa pemrograman untuk software agent yang dapat menangani masalah runtime tersebut.

Dari karakteristik di atas dapat disimpulkan bahwa bahasa pemrograman yang layak untuk mengimplementasikan software agent adalah sebagai berikut :

- Java
- Telescript
- Tcl/Tk, Safe-Tcl, Agent-Tcl

Referensi:

1. http://bebas.vlsm.org/v06/Kuliah/SistemOperasi/BUKU/SistemOperasi-4.X-1/ch11.html#c31101
2. http://bebas.vlsm.org/v06/Kuliah/SistemOperasi/2002/riene101/produk/Materi/Threads.html
3. Rofiq Siregar, Definisi Client/Server, http://rofiqsiregar.wordpress.com/2007/05/09/defenisi-clientserver/
4. http://bebas.vlsm.org/v06/Kuliah/SistemOperasi/BUKU/SistemOperasi-4.X-1/ch17s05.html
5. Rofiq Siregar, Model Client/Server, http://rofiqsiregar.wordpress.com/2007/05/29/model-clientserver/
6. Ayu Anggriani dkk., Tugas Kuliah Pengantar Sistem Terdistribusi, 2008.
7. Romi Satria Wahono, Pengantar Software Agent: Teori dan Aplikasi, http://ilmukomputer.com/2006/08/21/pengantar-**software-agent**/