BAB II
LANDASAN TEORI

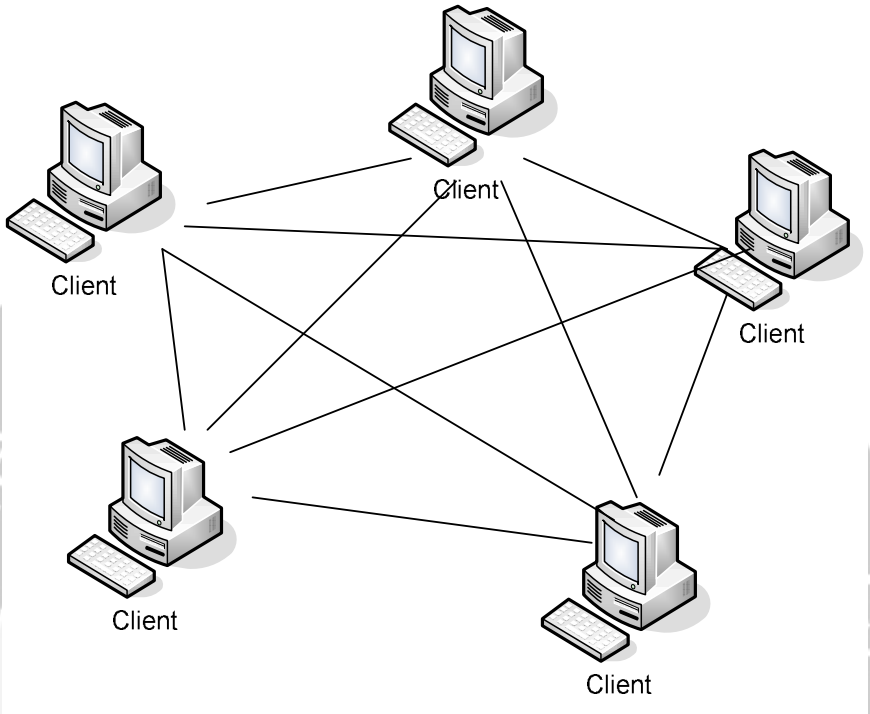
II.1 Konsep Jaringan Komputer

Jaringan komputer adalah sekelompok komputer yang dihubungkan dengan yang lainnnya menggunakan protokol komnuikasi melalui media transmisi atau media komnikasi sehingga dapat saling bertukar informasi, penggunaan bersama perangkat keras, seperti printer, harddisk, dan memberi layanan komunikasi antar pemakai (Agung, 1998). Dua buah komputer dikatakan terinterkoneksi bila keduanya dapat saling bertukar informasi. Bentuk koneksinya tidak perlu melalui kawat tembaga saja, tetapi dapat melalui serat optik, gelombang mikro, dan satelit kounikasi.

II.2 Model Jaringan komputer

II.2.1 Model Peer-to-Peer (P2P)

Definisi peer to peer adalah suatu teknologi *sharing* (pemakaian bersama) *resource* dan *service* antara satu komputer dan komputer lain. *Resource* disini bisa berupa memori, *CPU (Central Proccessing Unit)*, *disk storage*, informasi, dan sebagainya. Definisi lain yang lebih teknis tentang P2P adalah sistem komputerisasi *client-server* dimana suatu mesin (komputer) berfungsi sebagai *client* sekaligus server, sehingga memungkinkan komunikasi dan pertukaran resource antara 2 komputer secara langsung (*real time*).



Gambar II.1 Arsitektur Peer to Peer

Beberapa keuntungan pemakaian teknologi P2P :

a. Mengurangi beban kerja *server*.
b. Mengurangi biaya pengadministrasian *server* terpusat.
c. Kolaborasi antara 2 orang (pekerja) dalam satu tim bisa berjalan lebih baik dan cepat.

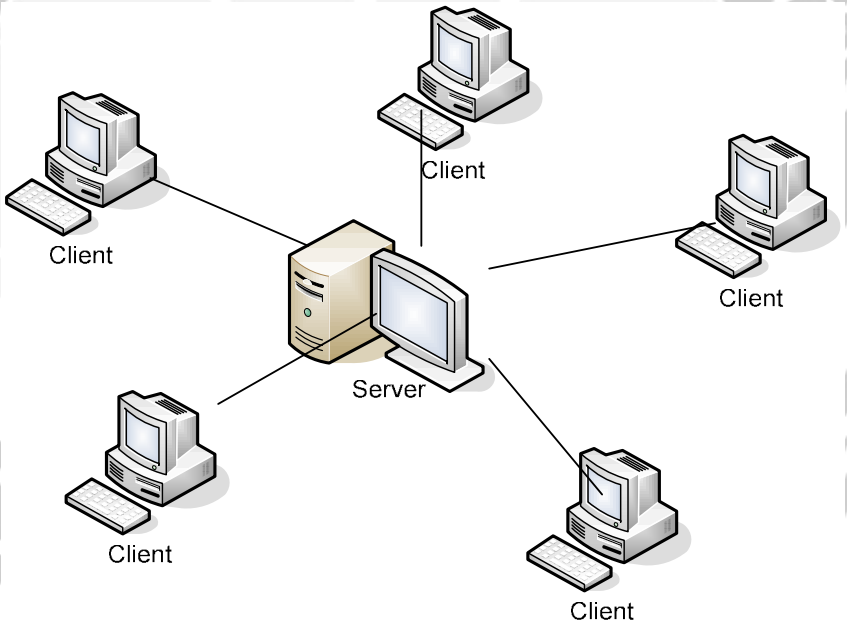
Saat ini P2P berkembang menjadi 3 kategori, yaitu:

a. *Instant messaging*: merupakan komunikasi antara komputer dalam bentuk pesan dan instant karena lebih berbentuk seperti *chatting* atau percakapan antara beberapa *user* dengan menggunakan media teks dan menggunakan P2P untuk jaringannya.

b. *File sharing*: merupakan komunikasi antara komputer untuk berbagi file atau data, seperti halnya pada *instant messaging*, maka file dapat langsung diterima oleh komputer lain tanpa melalui *server* tertentu.

c. *Distributed computing*: pengerahan beratus-ratus ataupun beribu-beribu komputer yang terhubung secara *peer to peer* untuk mengerjakan komputasi data yang kompleks dan besar.

II.2.2 Model Client-Server



Gambar II.2 Arsitektur Client Server

Pada model ini terdapat sebuah atau lebih komputer yang bertintak sebagai *server* dan yang lain sebagai *client*. Komunikasi pada model ini pada umumnya berbentuk pesan permintaan untuk melaksanakan pekerjaan dari *client* kepada *server*. Setelah *server*

melaksanakan tugasnya lalu hasilnya akan dikirim kembali ke *client*.

II.3 IPC (Inter-Process Communication)

Inter-Process Communication (IPC) atau Komunikasi antar proses adalah cara atau mekanisme pertukaran data antara satu proses dengan proses lainnya, baik itu proses yang berada di dalam komputer yang sama, atau komputer jarak jauh yang terhubung melalui jaringan. Secara Garis besar ada 2 macam IPC dasar:

a. *Shared Memory*

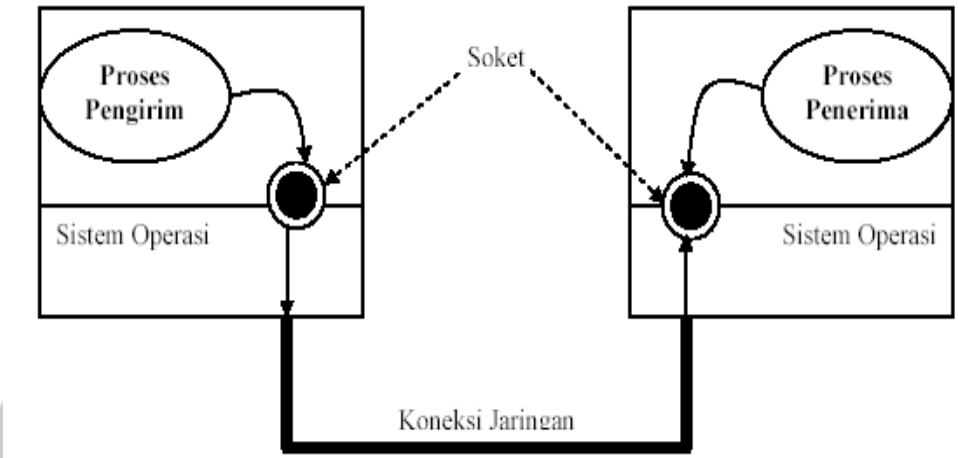
   Terutama pada sistem yang dimana proses yang saling berkomunikasi terletak pada satu node komputer

b. *Message Passing*

   Mekanisme yang dipakai untuk berkomunikasi antar proses yang terletak pada node komputer yang sama maupun berbeda.

II.4 Pemrograman Soket

Soket merupakan fasilitas IPC (*Inter-Process Communication*) untuk aplikasi jaringan. Soket yang telah dibuat dapat digunakan untuk menunggu koneksi atau untuk memulai suatu koneksi. Soket yang Digunakan server untuk menunggu koneksi yang datang disebut *passive socket*, sedangkan socket yang digunakan oleh *client* untuk memulai suatu koneksi disebut *active socket*. Perbedaan antara *passive* dan *active socket* tergantung pada bagaimana aplikasi menggunakannya.



Gambar II.3 Model IPC dengan Soket

Agar suatu soket dapat berkomunikasi dengan soket lainnya, maka soket butuh diberi suatu alamat unik sebagai identifikasi. Alamat soket terdiri atas Alamat IP dan Nomor Port. Contoh alamat soket adalah **192.168.29.30: 3000,** dimana nomer 3000 adalah nomor portnya. Alamat IP dapat menggunakan alamat Jaringan Lokal (LAN) maupun alamat internet. Jadi soket dapat digunakan untuk IPC pada LAN maupun Internet.

Nomor port dibutuhkan karena proses yang berjalan pada suatu komputer umumnya lebih dari satu. Sehingga dibutuhkan tambahan informasi sebagai identifikasi proses yang hendak dihubungi. Jika IP computer diibaratkan adalah nomor telepon suatu perusahaan, maka nomor port adalah nomor ekstensinya. Suatu proses yang hendak berkomunikasi dengan proses lain lewat mekanisme soket haruslah mengikatkan dirinya dengan salah satu port pada komputernya. Pengikatan diri ini disebut dengan *binding*.

II.5 Fungsi-Fungsi Dalam Soket

Fungsi-fungsi primer yang terdapat dalam soket, yaitu :

II.5.1 Fungsi Socket

Aplikasi menggunakan fungsi ini untuk membuat soket baru yang dapat digunakan dalam komunikasi jaringan.

II.5.2 Fungsi Connect

Setelah membuat soket, *client* memanggil fungsi connect untuk membuat koneksi dengan server. Fungsi ini memungkinkan client untuk menspesifikasikan *remote endpoint*, yang terdiri dari alamat IP dari *remote machine* dan nomor portnya. Setelah koneksi terbentuk, *client* dapat mentransfer data melaluinya.

II.5.3 Fungsi Send

Client dan *server* menggunakan fungsi *send* untuk mengirim data melalui koneksi TCP. *Client* biasanya menggunakan *send* untuk mengirimkan *request*, sementara *server* menggunakannnya untuk mengirimkan balasan dari request *client*.

II.5.4 Fungsi Receive

Client dan server menggunakan fungsi receive untuk menerima data dari koneksi TCP. Biasanya, setelah koneksi berdiri, server menggunakan fungsi ini untuk menerima *request* yang telah dikirimkan client dengan menggunakan fungsi *send*. Setelah mengirimkan request, *client* menggunakan fungsi *receive* untuk menerima balasan dari *server*.

II.5.5 Fungsi Closesocket

Setelah client atau server selesai menggunakan soket, fungsi *cloesocket* dipanggil untuk mengakhiri koneksi dan mendealokasi soket.

II.5.6 Fungsi Bind

Ketika soket pertama kali dibentuk, soket tersebut belum memiliki *endpoint address* (baik alamat lokal ataupun remote). Aplikasi memanggil fungsi *bind* untuk menspesifikasikan *local endpoint address* suatu soket. Untuk TCP/IP *local endpoint address* terdiri dari alamat IP dan nomor port.

II.5.7 Fungsi Listen

Connection-oriented server memanggil fungsi *listen* untuk meletakkan soket dalam mode pasif dan membuatnya siap untuk menerima koneksi yang datang. Sebagian besar server terdiri dari perulangan untuk menerima koneksi yang datang. Begitu suatu koneksi diterima soket server akan langsung menanganinya, kemudian perulangan akan berlanjut untuk menerima koneksi berikutnya.

II.5.8 Fungsi Accept

Untuk TCP Socket, setelah server memnaggil fungsi *socket* untuk membuat soket, *bind* untuk menspesifikasikan *local endpoint address*, dan *listen* untuk meletakkannya pada mode pasif, server memanggil fungsi *accept* untuk menspesifikasikan soket mana yang koneksinya harus diterima.

Fungsi *accept* membuat soket baru untuk setiap *request* koneksi baru dan mereturnkan deskriptor dari soket baru ke pemanggilnya. Setelah soket menerima suatu koneksi, server dapat menggunakan soket baru tersebut untuk mentrasnfer data. Setelah selesai menggunakan soket baru tersebut, server kemudian menutupnya dengan menggunakan fungsi *closesocket*.

II.6 Macam-Macam Komunikasi Soket

Secara umum ada dua macam komunikasi dengan menggunakan soket, yaitu komunikasi stream dan komunikasi datagram. Komunikasi stream sering juga disebut dengan komunikasi yang berorientasi koneksi (*Connection oriented communication*). Sedangkan Komunikasi datagram disebut juga dengan komunikasi tak berkoneksi (*connectionless communication*). Protokol standar untuk komunikasi stream dikenal dengan istilah TCP (*Transmission Control Protocol*), sedangkan standar protokol komunikasi datagram dikenal dengan UDP (*User Datagram Protocol)*.

Pada UDP, setiap kali suatu paket data dikirim, informasi soket pengirim dan alamat soket tujuan turut dikirimkan. Hal demikian tidak dibutuhkan oleh TCP, karena TCP akan membuat setup koneksi dengan soket tujuan terlebih dulu. Setelah koneksi terbentuk, tidak dibutuhkan mengirimkan informasi soket pengirim tiap kali data dikirimkan. Ini karena proses tujuan akan mengidentifikasi setiap data yang tiba pada soket tujuan sebagai data dari proses pengirim. Koneksi yang terbentuk pada TCP bersifat dua arah (*bidirectional*).

Perbedaan lain adalah UDP memiliki batasan ukuran datagram (paket data) yang dikirimkan sebesar 64 kb. Sedangkan TCP tidak memiliki batasan ini karena data-data dikirimkan sebagai aliran data (stream). Sesungguhnya TCP akan memecah data yang besar menjadi sejumlah paket data berukuran kecil dan diberi nomer urut. Pada sisi soket penerima, paket-paket data ini akan disimpan, diurutkan kembali, dan akhirnya digabungkan kembali menjadi data besar.

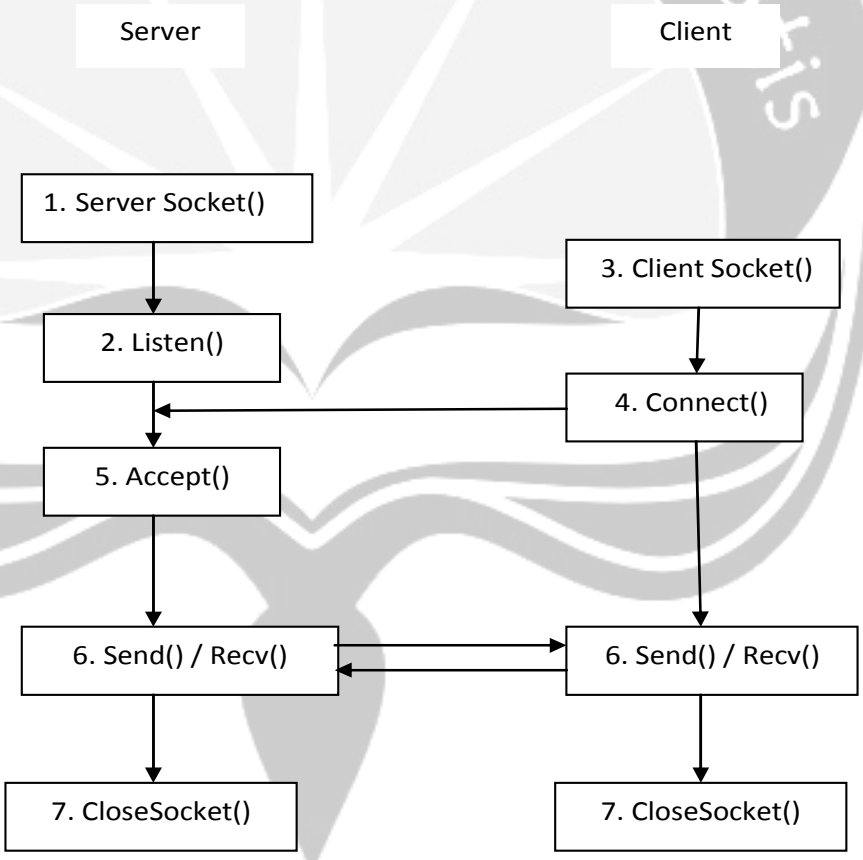
Perbedaan lain adalah UDP merupakan protocol yang *unreliable* (tidak handal). Ketika paket data dikirimkan, UDP tidak mengecek kembali apakah data yang dikirim sampai tujuan. Jadi dengan UDP tidak ada kepastian bagi sisi pengirim bahwa datanya sudah sampai ke tujuan dengan keadaan baik. Sebaliknya TCP adalah protocol yang *reliable* yang senantiasa menunggu konfirmasi dari pihak soket penerima, dan kalau perlu paket data yang hilang akan dikirimkan kembali. Konsekuensinya adalah TCP menimbulkan overhead lalulintas jaringan lebih tinggi dibanding UDP.

II.7 Model Aplikasi Client Server

Model aplikasi yang menggunakan komunikasi soket dengan protokol TCP digambarkan pada gambar II.4. Obyek soket pada sisi client dan server berbeda sedikit. Pada sisi aplikasi server, suatu soket server dibentuk (1) dan melakukan operasi *listen (2)*. Operasi ini pada intinya menunggu permintaan koneksi dari sisi client. Sedangkan pada sisi client, dibentuk suatu soket biasa.

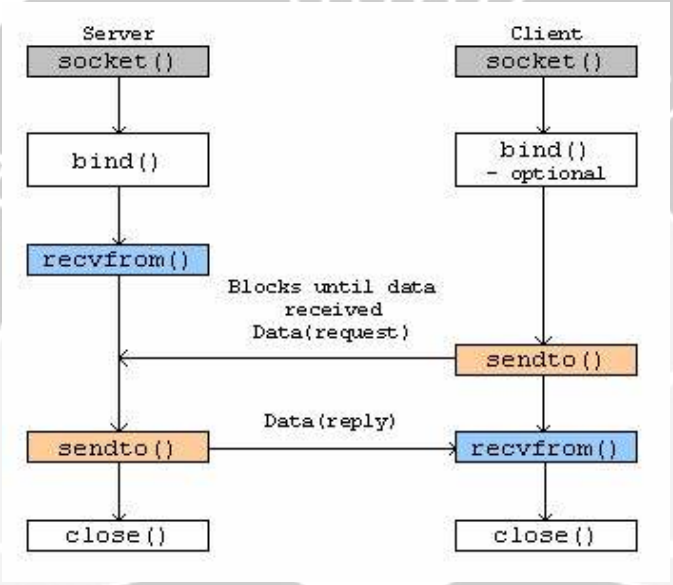
Pada sisi client (3), akan membentuk soket client. Kemudian (4) membentuk koneksi ke server, informasi alamat soket server dilewatkan sebagai argumen. Server akan menerima koneksi dari client (5). Server akan membuat suatu soket biasa. Soket ini yang akan berkomunikasi dengan soket pada sisi client.

Setelah tercipta koneksi antara client dan server, maka keduanya dapat saling bertukar pesan (6). Salah satu atau keduanya kemudian dapat mengakhiri komunikasi dengan menutup soket (7).



Gambar II.4 Model Aplikasi Client/Server pada protokol TCP

Untuk protokol UDP, perbedaanya adalah soket di sisi server sama dengan soket di sisi client, dan tidak ada operasi listen pada sisi server. Kemudian saat paket data dikirimkan, alamat soket penerima harus disertakan sebagai argumen.

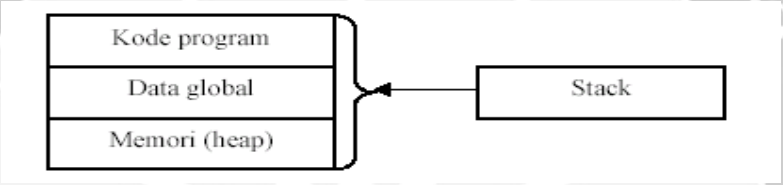


Gambar II.5 Model Aplikasi Client/Server protokol UDP

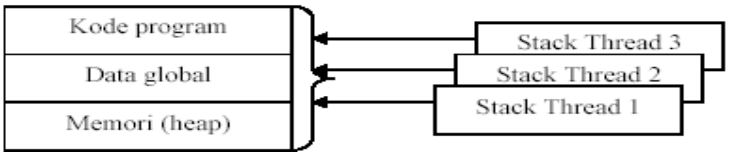
II.8 Konsep Pemrograman Multithreaded

Program dengan satu alur eksekusi disebut dengan program sekuensial. Program yang demikian hanya memiliki satu titik eksekusi pada tiap saat. Sedangkan program yang memiliki lebih dari satu alur eksekusi pada tiap saat disebut dengan program konkuren. Tiap unit alur eksekusi dari program konkuren disebut dengan *thread*. *Thread* dapat menciptakan thread baru lainnya ataupun membunuh thread yang sudah ada. *Thread* memberikan ilusi adanya sejumlah aktivitas yang berjalan pada saat bersamaan.

Namun bedanya dengan sejumlah proses yang berjalan pada sistem operasi, thread-thread yang berjalan pada satu aplikasi memiliki ruang alamat yang sama. Artinya thread thread tersebut memiliki akses bersama ke kode program, memori dan data global yang sama seperti pada gambar 2. Selain struktur data bersama yang dapat diakses bersama, thread juga memiliki struktur data private atau khusus (variabel lokal *thread*) yang disimpan pada *stack* thread dan hanya dapat diakses oleh thread bersangkutan.



Gambar II.6 Program sekuensial



Gambar II.7 Program Konkuren

Keuntungan menggunakan sejumlah thread dalam program antara lain: kemampuan melakukan sejumlah tugas secara simultan dan tak bergantung satu sama lain. Misalkan satu thread melakukan penggambaran animasi secara background, sedangkan thread lain menangani inputan user lewat keyboard. Selain itu thread dapat meningkatkan *throughput* aplikasi, memperbaiki waktu tanggapan (*responsiveness*) aplikasi dan kemampuan untuk menggunakan sumber daya sistem secara efisien.

II.9 File Image Virtual Machine

File image virtual machine merupakan suatu kontainer yang menampung informasi penting untuk mejalankan virtual machine. File ini berisi seluruh informasi yang dibutuhkan untuk menjalankan komputer dalam virtual machine dengan aplikasi-aplikasi yang terinstal didalamnya. Tidak seperti file-file biasa, untuk memperoleh informasi-informasi penting tersebut file image virtual machine harus "dieksekusi". Persyaratan daya tampung untuk file image virtual machine biasanya lebih besar dibanding dengan file-file lainnnya, karena file image ini pada umumnya berukuran besar (Gbs).