Rangkuman

Network Programming

Diusulkan oleh:

Rifky Ajie Nugraha

2001595815

UNIVERSITAS BINA NUSANTARA

JAKARTA

2018

A. Socket

Socket adalah sebuah mekanisme pada komputer yang memperbolehkan komunikasi antara proses-proses yang berbeda dalam komputer maupun antar komputer yang berbeda. **Lebih singkat, socket merupakan mekanisme komunikasi antar komputer**.

Socket biasa dipakai pada aplikasi berbasis *Client-Server*. Server merupakan aplikasi yang digunakan untuk memproses permintaan client. Beberapa contoh *Protocol* yang memerlukan socket adalah :

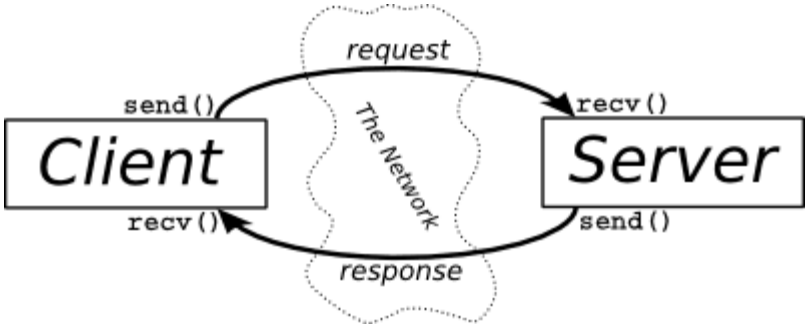
- TCP (Transmission Control Protocol)

- FTP (File Transfer Protocol)

- SMTP (Simple Mail Transfer Protocol)

- POP3 (Post Office Protocol version 3)

Berikut adalah gambaaran sederhana dari kerja aplikasi server-client :



https://therighttutorial.wordpress.com/2014/06/09/multi-client-server-chat-application-using-socket-programming-tcp/

send() dan *recv()* merupakan bagian dari socket, fungsi mereka adalah sebagai jalur komunikasi antar aplikasi untuk menerima dan mengirim data melalui jaringan.

Jenis – Jenis Socket

Socket memiliki berbagai macam jenis dan biasanya dibedakan dengan protokolnya, berikut beberapa jenis socket :

- *Stream Socket*

  Socket ini digunakan pada protokol TCP untuk transmisi data, data tsb dikirimkan secara berurutan dan pengirimannya terjamin.

- *Datagram Socket*

  Socket ini digunakan pada protokol UDP untuk transmisi data, UDP mengirimkan data hanya dengan mengandalkan informasi alamat tujuan sehingga pengirimannya tidak terjamin.

- *Raw Socket*

  Socket ini dapat didefinisikan sesuai dengan protokol yang dipakai, dan biasa digunakan pada orang yang mengembangkan protokol baru

B. Network System

Network System merupakan sistem yang menyediakan layanan jaringan untuk keperluan komunikasi dan transfer data antar komputer.

Network System Elements

- Computing Terminals >>> PC, Workstation, etc.
- Communication Links >>> LAN, WAN, Modem, etc.
- Communication Protocols >>> TCP, UDP, etc.
- System Software >>> OS, Protocol Stack, etc.
- Application Software >>> Web, FTP, etc.

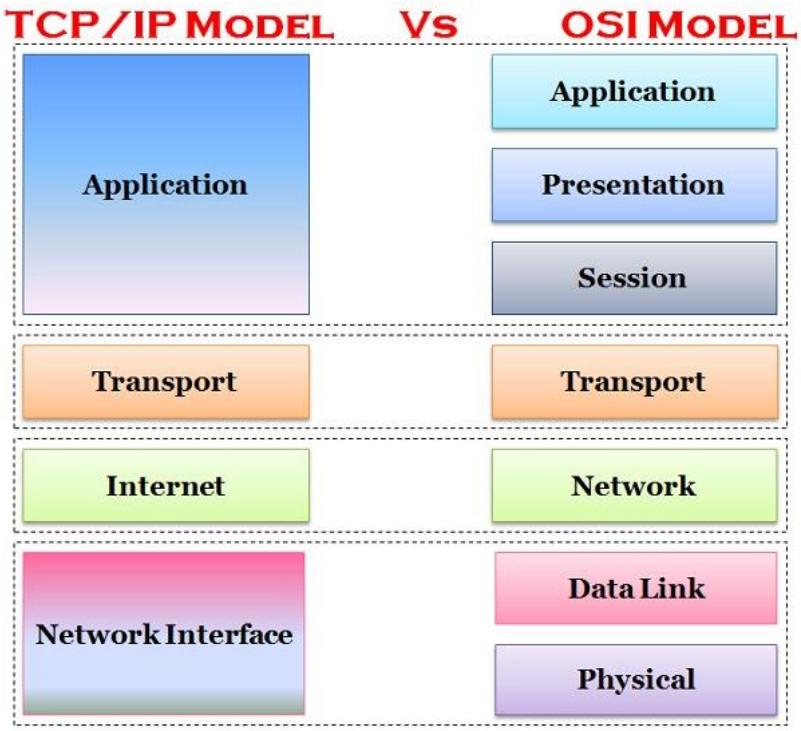
OSI Model & TCP/IP Model

OSI atau *Open System Interconnection* merupakan model komunikasi yang dibuat untuk jaringan komputer. OSI model memiliki 7 lapisan (*layer*), berikut beserta fungsinya :

LAYER	FUNGSI	CONTOH
Application (7)	Menunjang aplikasi untuk berkomunikasi melalui jaringan	SMTP
Presentation (6)	Memformat data sehingga dapat dikenali oleh penerima	JPG, GIF, HTTPS, SSL, TLS
Session (5)	Membentuk koneksi, kemudian memutuskannya ketika seluruh data telah terkirim	NetBIOS, PPTP
Transport (4)	Mengatur flow control, acknowledgment dan mengirim ulang data jika diperlukan	TCP, UDP
Network (3)	Menambahkan alamat jaringan pada paket	Router, Layer 3 Switch
Data Link (2)	Menambahkan MAC address pada paket	Switch
Physical (1)	Mengirimkan data melalui media transmisi	Hub, NIC, Kabel

NESABAMEDIA.COM

https://www.nesabamedia.com/pengertian-osi-layer/



https://techdifferences.com/difference-between-tcp-ip-and-osi-model.html

Untuk TCP/IP sendiri modelnya tidak berbeda jauh dengan OSI. TCP/IP hanya memiliki 4 layer, layer *Session, Presentation, & Application* pada OSI digabung menjadi *Application* pada TCP/IP. Walaupun demikian, fungsi dasar dari keduanya masih tetap sama.

C. Socket Programming

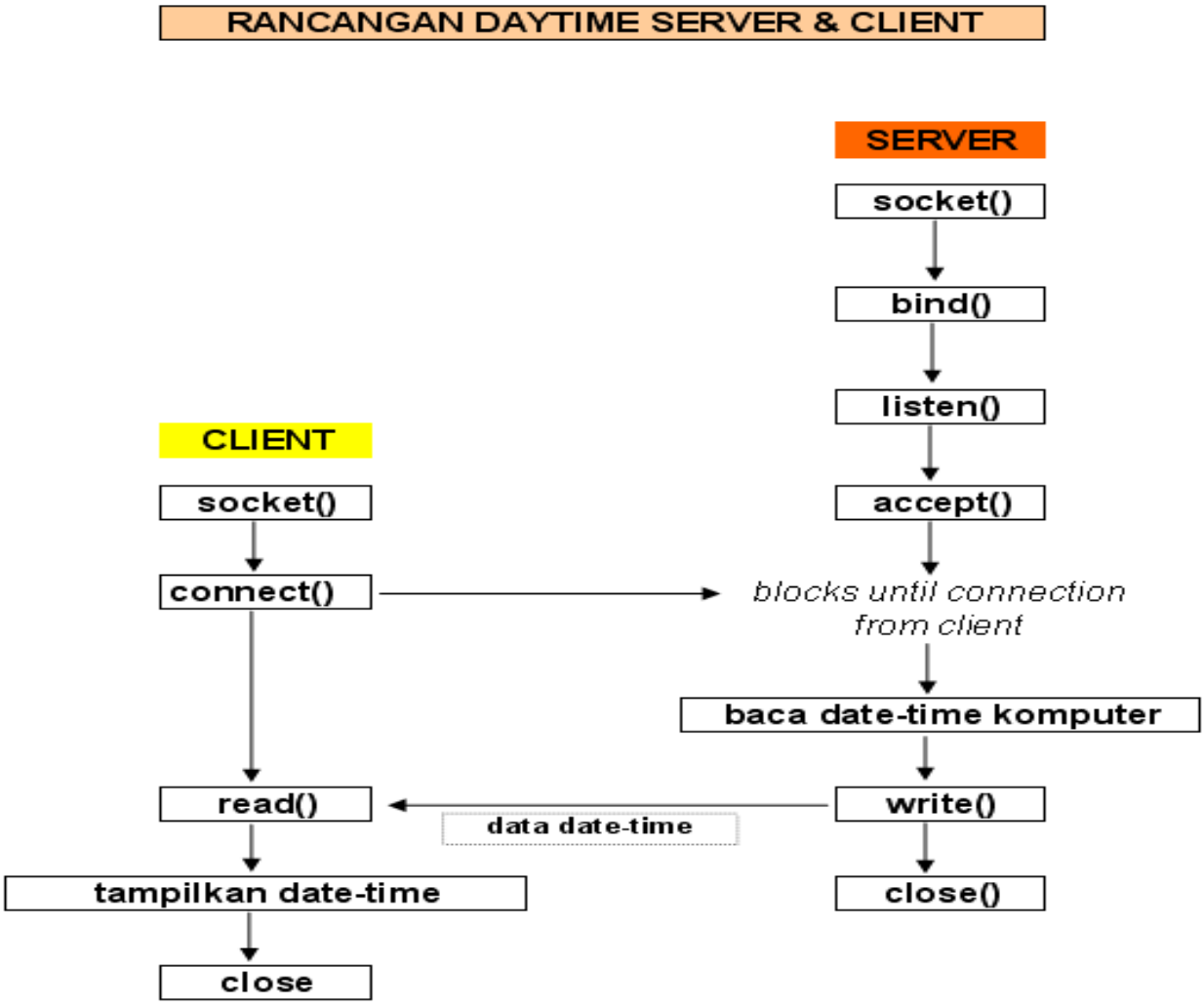
Socket Programming merupakan pemrogramman yang melibatkan komunikasi antar proses/komputer, salah satu contoh socket programming adalah *server-client.*

Konsep Dasar Server-Client

Server-Client merupakan aplikasi yang membuat komputer/proses berinteraksi stau sama lain. *Client* merupakan aplikasi yang tugasnya mengirim permintaan *user* ke *server* lalu permintaan tersebut akan diproses oleh *Server.*

Protokol yang biasa digunakan untuk server-client adalah TCP dan UDP, keduanya memiliki tujuan yang sama namun jalan kerjanya yang berbeda. TCP lebih diandalkan untuk pengiriman data seperti *Download* dll. dikarenakan pengirimannya yang lebih terjamin sehingga mengurangi resiko *data corruption.* Sedangkan UDP lebih digunakan untuk streaming karena pengirimannya yang bisa lebih cepat walaupun tidak seakurat TCP.

Berikut contoh alur program server & client TCP untuk meminta informasi waktu :



Slide Bimay : Elementary Socket

Server :

- *socket()*

  Membuat socket yang nantinya diperlukan untuk komunikasi.

- *bind()*

  Menggabung/menyatukan alamat, port dengan socket yang telah dibuat

- *listen()*

  Menungguu koneksi dari *client.*

- *accept()*

  Menerima apabila ada koneksi yang masuk.

- *write()*

  Menulis dan mengirim data yang akan dikirim pada jaringan.

- *close()*

  Menutup socket yang sudah dipakai

Client :

- *socket()*

  Membuat socket yang nantinya diperlukan untuk komunikasi.

- *connect()*

  Menghubung ke aserver / alamat tujuan.

- *read()*

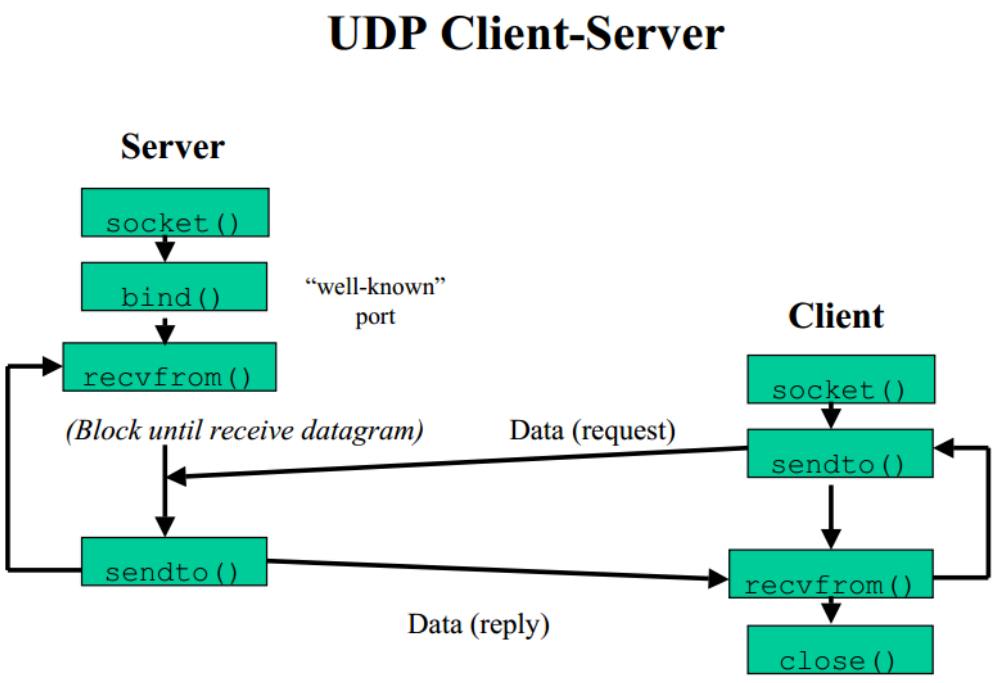
  Membaca buffer yang disediakan pada jaringan bila ada data yang masuk.

- *close()*

  Menutup socket yang sudah dipakai.

Pada alur TCP, server harus membuat koneksi terlebih dahulu dengan client baru berinteraksi, hal ini bertujuan untuk menjaga integritas data. Proses ini juga bisa dinamakan dengan ***Two-Way Handshake*** (atau *Three-Way Handshake*(lebih aman))**.**

Berikut kerangka pada *Server & Client* UDP :



https://stackoverflow.com/questions/23068905/is-bind-necessary-if-i-want-to-receive-data-from-a-server-in-c-udp

Server&Client :

- *recvfrom()*

  Menerima data yang menuju ke mesin itu sendiri

- *sendto()*

Mengirim ke alamat tujuan

Bila dilihat UDP lebih singkat dibanding TCP, hal ini memang demikian karena **UDP hanya memerlukan informasi tujuan (alamat & port) untuk mengirim data sehingga prosesnya lebih cepat namun bermasalah pada integritas datanya.**

Bahasa pemprograman yang dapat digunakan dalam socket programming ada banyak, misalnya seperti C/C++, Java, Python, dan bahkan *php* yang merupakan bahasa pemprograman untuk *web* dapat digunakan untuk socket programming.

C/C++ Socket Programming

Berikut contoh kodingan TCP *Server-Client* pada C/C++ untuk mengirim pesan :

dikutip dari : https://www.binarytides.com/server-client-example-c-sockets-linux/

Server==========

```
/*

    C socket server example
*/

#include<stdio.h>
#include<string.h>    //strlen
#include<sys/socket.h>
#include<arpa/inet.h> //inet_addr
#include<unistd.h>    //write

int main(int argc , char *argv[])
{
    int socket_desc , client_sock , c , read_size;
    struct sockaddr_in server , client;
    char client_message[2000];

    //Create socket
    socket_desc = socket(AF_INET , SOCK_STREAM , 0);
    if (socket_desc == -1)
    {
        printf("Could not create socket");
    }
    puts("Socket created");

    //Prepare the sockaddr_in structure
    server.sin_family = AF_INET;
    server.sin_addr.s_addr = INADDR_ANY;
    server.sin_port = htons( 8888 );

    //Bind
    if( bind(socket_desc,(struct sockaddr *)&server , sizeof(server)) < 0)
    {
        //print the error message
        perror("bind failed. Error");
```

```
        return 1;
    }
    puts("bind done");

    //Listen
    listen(socket_desc , 3);

    //Accept and incoming connection
    puts("Waiting for incoming connections...");
    c = sizeof(struct sockaddr_in);

    //accept connection from an incoming client
    client_sock = accept(socket_desc, (struct sockaddr *)&client,
(socklen_t*)&c);
    if (client_sock < 0)
    {
        perror("accept failed");
        return 1;
    }
    puts("Connection accepted");

    //Receive a message from client
    while( (read_size = recv(client_sock , client_message , 2000 , 0)) > 0
)
    {
        //Send the message back to client
        write(client_sock , client_message , strlen(client_message));
    }

    if(read_size == 0)
    {
        puts("Client disconnected");
        fflush(stdout);
    }
    else if(read_size == -1)
    {
        perror("recv failed");
    }

    return 0;
}
```

==========

Client==========

```
/*
    C ECHO client example using sockets
*/
#include<stdio.h> //printf
#include<string.h>    //strlen
#include<sys/socket.h>    //socket
#include<arpa/inet.h> //inet_addr

int main(int argc , char *argv[])
{
    int sock;
    struct sockaddr_in server;
    char message[1000] , server_reply[2000];

    //Create socket
```

```
    sock = socket(AF_INET , SOCK_STREAM , 0);
    if (sock == -1)
    {
        printf("Could not create socket");
    }
    puts("Socket created");

    server.sin_addr.s_addr = inet_addr("127.0.0.1");
    server.sin_family = AF_INET;
    server.sin_port = htons( 8888 );

    //Connect to remote server
    if (connect(sock , (struct sockaddr *)&server , sizeof(server)) < 0)
    {
        perror("connect failed. Error");
        return 1;
    }

    puts("Connected\n");

    //keep communicating with server
    while(1)
    {
        printf("Enter message : ");
        scanf("%s" , message);

        //Send some data
        if( send(sock , message , strlen(message) , 0) < 0)
        {
            puts("Send failed");
            return 1;
        }

        //Receive a reply from the server
        if( recv(sock , server_reply , 2000 , 0) < 0)
        {
            puts("recv failed");
            break;
        }

        puts("Server reply :");
        puts(server_reply);
    }

    close(sock);
    return 0;
}
```

==========

Java Socket Programming

Berikut contoh kodingan *Server-Client* pada Java untuk mengirim pesan :

dikutip dari : https://stackoverflow.com/questions/2165006/simple-java-client-server-program

Server==========

```
import java.io.*;
import java.net.*;

public class TCPClient {
   public static void main(String argv[]) throws Exception {
      String sentence;
      String modifiedSentence;
      BufferedReader inFromUser = new BufferedReader(new InputStreamReader(System.in));

      Socket clientSocket = new Socket("localhost", 6789);
      DataOutputStream outToServer = new DataOutputStream(clientSocket.getOutputStream());
      BufferedReader inFromServer = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));

      sentence = inFromUser.readLine();
      outToServer.writeBytes(sentence + '\n');
      modifiedSentence = inFromServer.readLine();
      System.out.println(modifiedSentence);
      clientSocket.close();
   }
}
```

==========

Client==========

```
import java.io.*;
import java.net.*;

public class TCPServer {
   public static void main(String args[]) throws Exception {
      String clientSentence;
      String capitalizedSentence;
      ServerSocket welcomeSocket = new ServerSocket(6789);

      while(true) {
         Socket connectionSocket = welcomeSocket.accept();
         BufferedReader inFromClient = new BufferedReader(new
InputStreamReader(connectionSocket.getInputStream()));
         DataOutputStream outToClient = new DataOutputStream(connectionSocket.getOutputStream());
         clientSentence = inFromClient.readLine();
         capitalizedSentence = clientSentence.toUpperCase() + '\n';
         outToClient.writeBytes(capitalizedSentence);
      }
   }
}
```

==========

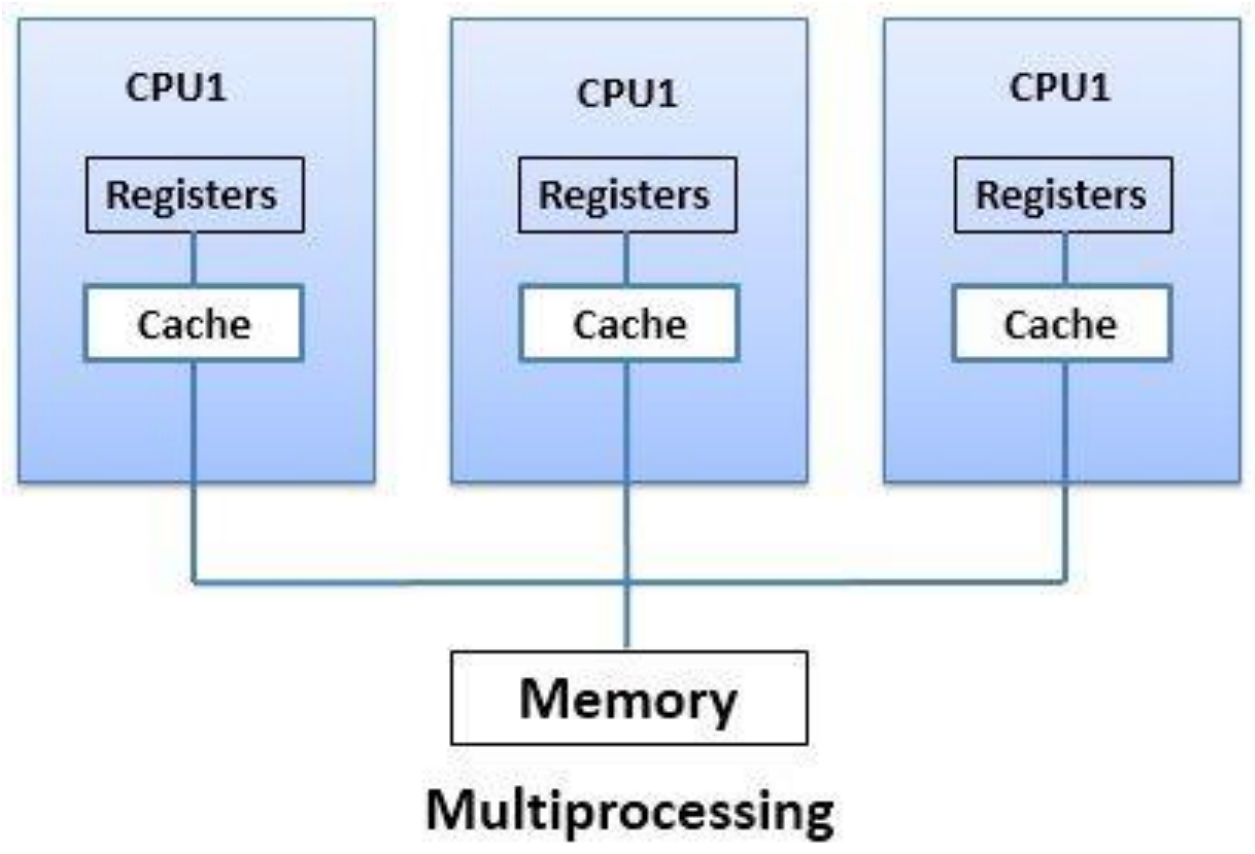
Lebih Dalam Mengenai Server

Dengan adanya komunikasi jaringan, pengiriman informasi menjadi lebih cepat dan mudah namun apa jadinya bila pengguna perangkat tersebut semakin banyak, apakah performa kerja pada program tersebut terganggu ? apakah server mampu menangani jumlah request dari client yang sangat banyak ? apakah server dapat mengatasinya dengan efisien ? have science gone too far ?.

Tentu masalah diatas sudah diperhatikan terlebih dahulu sehingga muncul beberapa teknik-teknik server yang lebih baik. Berikut adalah contohnya :

1. Multiprocessing

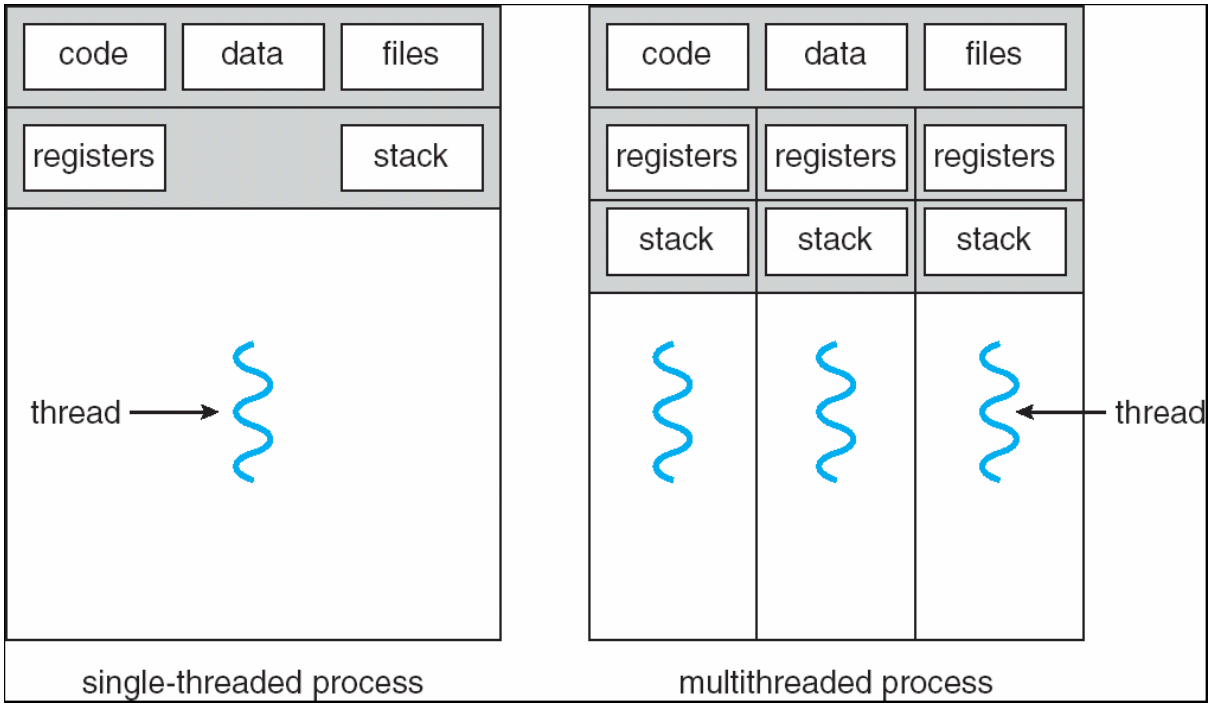
Multiprocessing merupakan teknik pengeksekusian program secara serentak, sehingga layanan yang diberikan lebih besar dibandingkan penjallanan program dalam satu waktu, program yang sama dijalankan sebanyak *n* secara serentak dalam satu waktu.



https://techdifferences.com/difference-between-multiprocessing-and-multithreading.html

2. Multithreading

Multithreading merupakan teknik pada program untuk menjalankan sebagian dari program didalam program itu sendiri secara banyak tanpa mengganggu aktifitasnya yang lain sehingga memberikan efisiensi yang lebih baik dan layanan yang lebih banyak.



https://tintakopi.wordpress.com/2011/10/24/thread-multithread/

3. I/O Multiplexing

I/O Multiplexing merupakan kemampuan server untuk memberitahu *kernel* dalam ketersediaan *I/O Condition* atau proses *input/output*. Ada beberapa macam *I/O model* :

- *Blocking I/O Model*

Merupakan model default pada tiap socket. Pada model ini I/O akan memblok input ketika sedang menunggu request.

- Non-*Blocking I/O Model*

Pada model ini I/O akan diberi *sleep* ketika tidak ada input yang masuk.

- *I/O Multiplexing Model*

Pada model ini program menggunakan *threading* sehingga proses input tidak terhalang apabila program menunggu request.

- *Signal Driven/Synchronus I/O Model*

Pada model ini memanfaatkan sinyal untuk memberitahu program bila ada data yang siap untuk dibaca maka program akan menerima dan membaca data tsb.

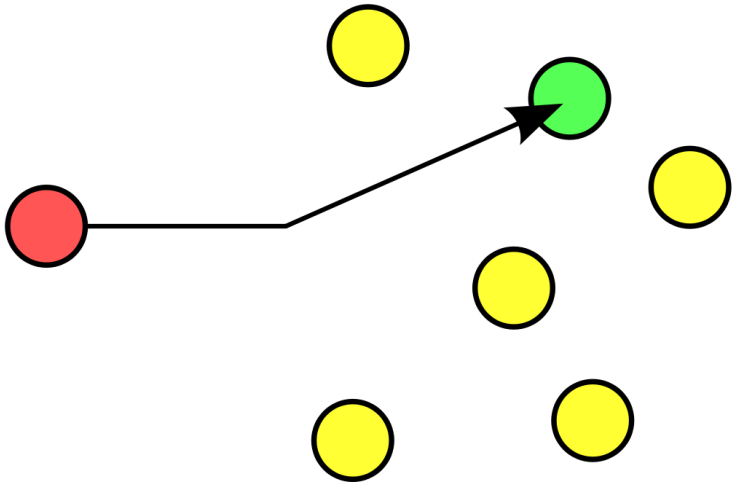
- *Asynchronus I/O Model*

Pada model ini memanfaatkan data akan dibaca terlebih dahulu baru mengirimkan sinyal

D. Network Programming Technique (Unicast, Multicast, Broadcast)

Unicast

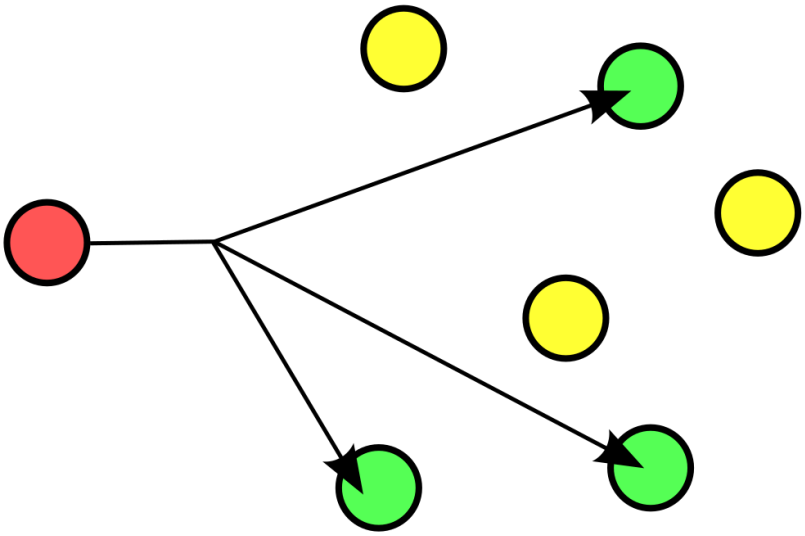
Unicast merupakan teknik pengiriman data dimana sumber menentukan 1 tujuan pengiriman dari data tersebut.



https://en.wikipedia.org/wiki/Unicast

Multicast

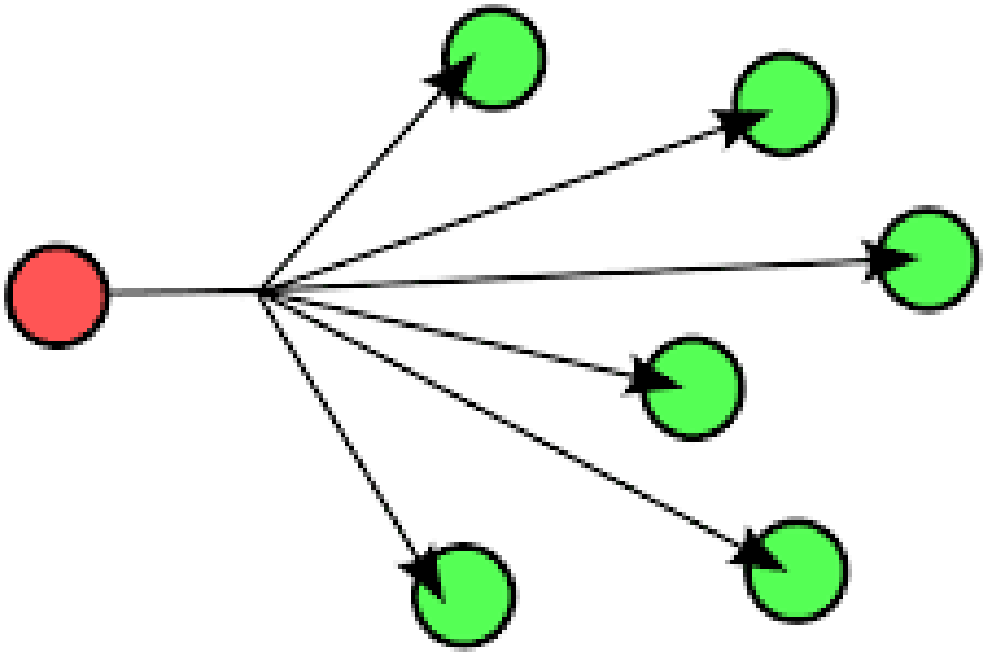
Multicast merupakan teknik pengiriman data dimana sumber menentukan beberapa tujuan pengirimannya.



https://en.wikipedia.org/wiki/Multicast

Broadcast

Broadcast merupakan teknik pengiriman data dimana semua perangkat yang ada pada satu area jaringan dengan source akan meneriman data tersebut.



---

http://tofanworld.blogspot.com/2015/09/pengertian-tentang-unicast-multicast.html

Pada paket broadcast, nomor pada paketnya tertulis ff:ff:ff:ff:ff:ff: .