PRAKTIKUM 9
STREAM SOCKET PROGRAMMING

I. Tujuan

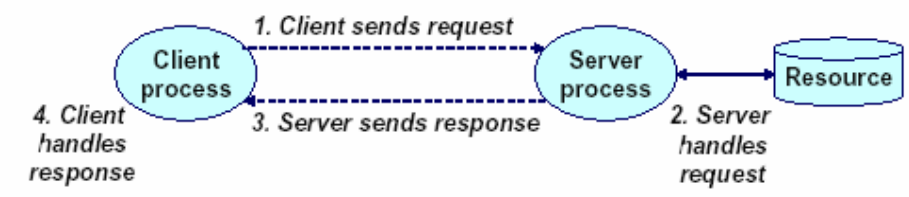
1. Mahasiswa memahami konsep aplikasi client server di jaringan.
2. Mahasiswa memahami konsep pemprograman socket.
3. Mahasiswa memahami jenis – jenis socket programming
4. Mahasiswa mampu membangun program socket sederhana

II. Peralatan Yang Dibutuhkan

1. Beberapa komputer yang berfungsi sebagai server.
2. Beberapa komputer yang berfungsi sebagai *client*.
3. *Hub/switch* sebagai penghubung jaringan.
4. Kabel jaringan secukupnya.

III. Dasar Teori

Setiap aplikasi di jaringan, transaksinya didasarkan pada konsep *client-server*. Sebuah *server* dan sebuah atau beberapa *client* yang meminta/*request* pelayanan ke server. Fungsi server sebagai pengatur *resource* yang ada, yang menyediakan pelayanan dengan memanfaatkan resource yang untuk kebutuhan client. Proses ini (proses *client-server*) bisa dijalankan pada sebuah komputer (komputer tunggal) atau bisa juga satu komputer berfungsi sebagai server dan sebuah atau beberapa komputer berfungsi sebagai client.

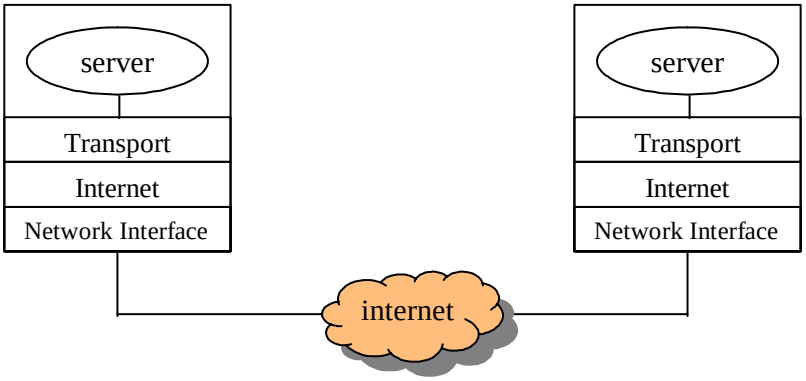


Gambar 1 Transaksi Client – server

TCP/IP protocol untuk interaksi Client-server

Untuk proses interaksi client-server dapat digunakan banyak protokol, tetapi untuk program ini akan digunakan protokol TCP/IP. Pertimbangannya adalah karena protokol TCP/IP adalah protokol standard yang paling banyak digunakan untuk komunikasi data di internet.

Program client-server menggunakan transport protocol untuk berkomunikasi seperti terlihat pada gambar di bawah ini :



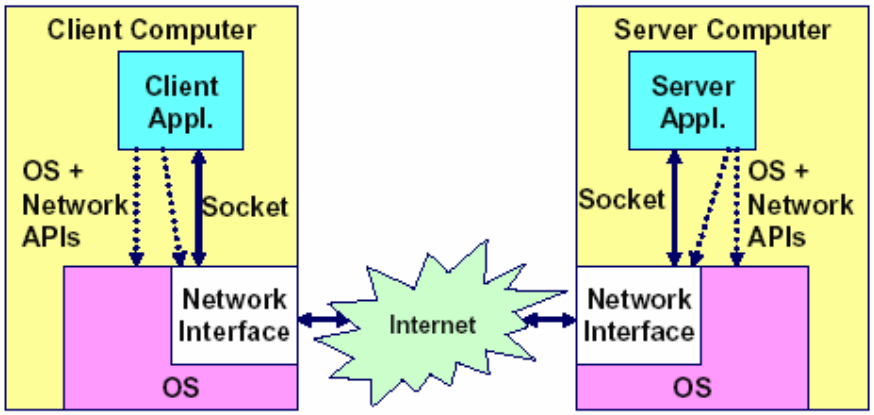
Gambar 2. Komunikasi client-server menggunakan TCP/IP

Aplikasi *client-server* menggunakan protokol *tranport* untuk saling berinteraksi. Ketika proses interaksi terjadi, suatu aplikasi harus memberikan informasi-informasi secara detail tentang :

- Informasi tentang apakah dia *client* atau *server*.
- Pengirim memberikan informasi tentang data yang dikirim.
- Penerima memberikan informasi tentang dimana data diletakkan, dll.

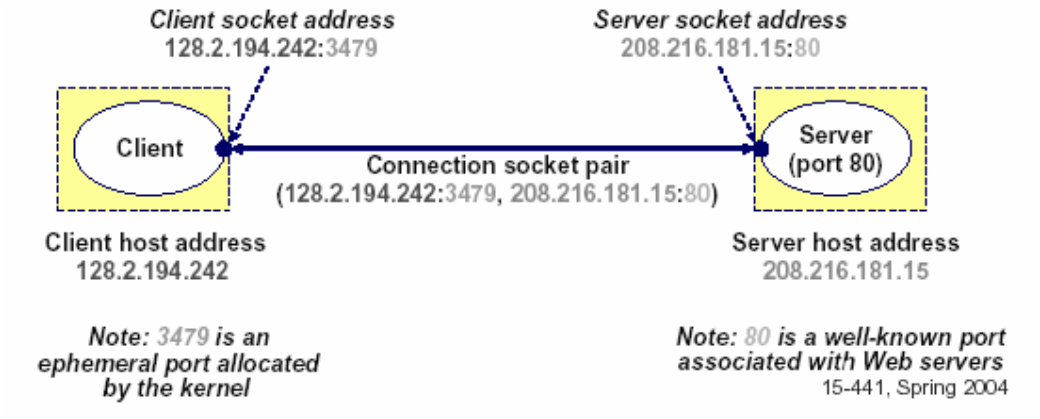
Antarmuka (*interface*) antara program aplikasi dengan protokol komunikasi pada suatu sistem operasi disebut *Application Program Interface (API). API* didefinisikan sebagai suatu kumpulan instruksi yang mendukung proses interaksi antara suatu perangkat lunak dengan suatu protokol yang digunakan.

Pada mesin keluarga *Linux, socket* terintegrasi dengan *I/O* sehingga aplikasi yang berkomunikasi dengan *socket*, cara kerjanya sama dengan suatu aplikasi yang mengakses peralatan *I/O*. Oleh karena itu untuk memahami cara kerja socket pada *Linux*, sebelumnya harus juga memahami fasilitas *I/O* pada *Linux*.



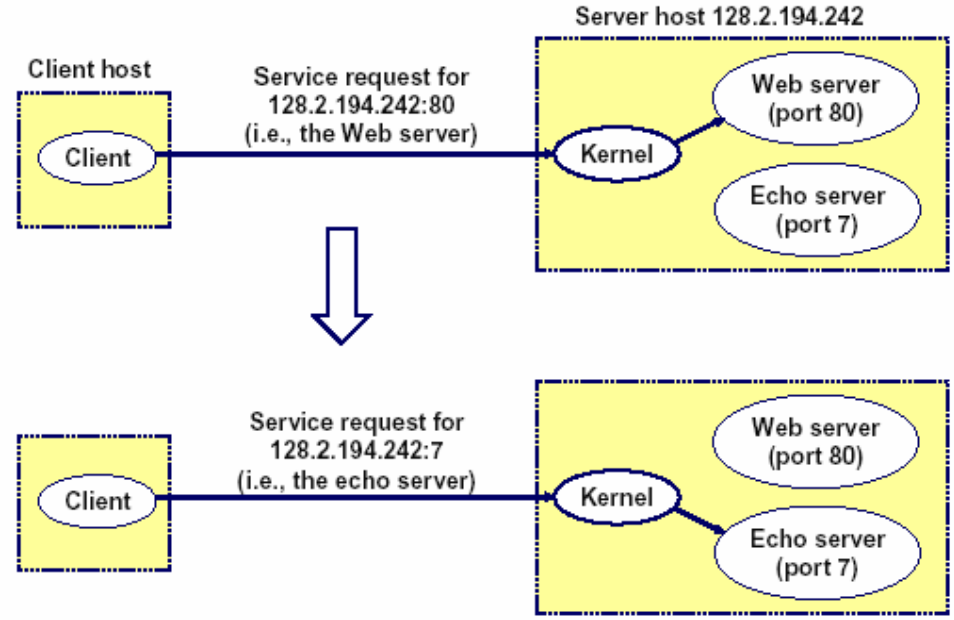
Gambar 3 Aplikasi Jaringan

Pada saat suatu aplikasi berkomunikasi, awalnya aplikasi membuat *socket* baru, maka pada aplikasi tersebut akan diberikan nomer yang digunakan sebagai referensi *socket*. Jika ada suatu sistem yang menggunakan nomer referensi *socket* tersebut, maka akan terjalin suatu jaringan komunikasi antar komputer sebaik transfer data lokal.



Untuk berkomunikasi dengan *server*, *client* harus tahu nomor *IP server* begitu juga nomor *port* yang dituju, nomor port menunjukkan service yang dijalankan. Contoh port 23 untuk *Telnet Server*, port 25 untuk *Mail Server* dan port 80 untuk *Web Server*. Dalam hal ini aplikasi di client sudah mengetahui port yang akan dituju. Contoh program aplikasi di client yang meminta service di server ada;ah ftp, telnet, ssh. Untuk melihat service bisa dilihat pada file */etc/services*.

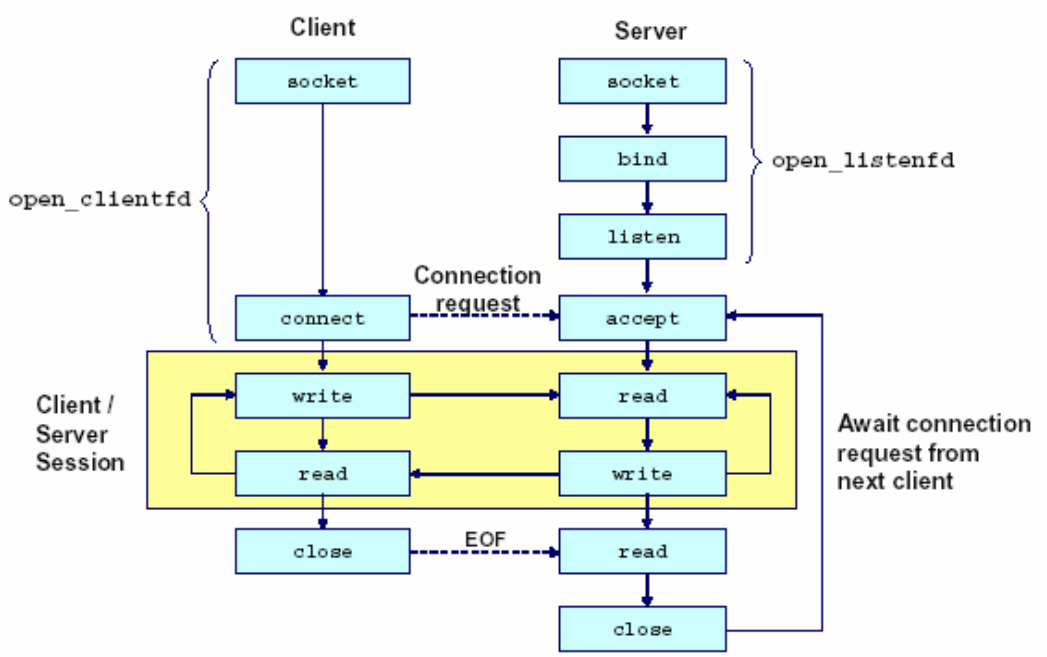
Program yang berjalan di server, akan berjalan sepanjang waktu (disebut sebagai daemon) sampai mesin/service dimatikan, menunggu request dari *client* sesuai service yang diminta.



Gambar 4 Menggunakan Port untuk identifikasi *service*

Jadi Socket adalah mekanisme komunikasi yang memungkinkan terjadinya pertukaran data antar program atau proses baik dalam satu mesin maupun antar mesin. Gaya pemrograman soket sendiri berawal dari sistem Unix BSD yang terkenal dengan kepeloporannya pada bidang penanganan jaringan, sehingga sering disebut BSD Socket. Socket pertama kali diperkenalkan di sistem Unix BSD versi 4.2 tahun 1983 sebagai kelanjutan dari implementasi protokol TCP/IP yang muncul pertama kali pada sistem Unix BSD 4.1 pada akhir 1981. Hampir setiap varian Unix dan Linux mengadopsi BSD Socket.

Linux menggunakan paradigma *open-read-write-close*. Sebagai contoh, suatu aplikasi pertama harus memanggil *open* untuk menyiapkan *file* yang akan diakses. Kemudian aplikasi tersebut memanggil *read* atau *write* untuk membaca data dari pada *file* atau menuliskan data ke *file*. Setelah itu *close* dijalankan untuk mengakhiri aplikasi yang digunakan. Interface soket dalam berkomunikasi bisa dilihat pada gambar 5 berikut :



Gambar 5 Ilustrasi Interface socket

Di dalam kotak menunjukkan system call/function yang dibutuhkan untuk koneksi/komunikasi, misal socket(), bind(), listen(), connect(), dll. Secara garis besar langkah – langkah yang dilakukan pada client dan server adalah sebagai berikut :

1. Langkah – langkah dasar di client :
   a. Membuka koneksi client ke server, yang di dalamnya adalah :
      - ✓ Membuat socket dengan perintah socket()
      - ✓ melakukan pengalamatan ke server.
      - ✓ Menghubungi server dengan *connect()*
   b. Melakukan komunikasi (mengirim dan menerima data), dengan menggunakan perintah *write()* dan *read()*
   c. Menutup hubungan dengan perintah *close()*;

2. Langkah – langkah dasar di server :
   a. Membuat socket dengan perintah *socket()*
   b. Mengikatkan socket kepada sebuah alamat network dengan perintah *bind()*
   c. Menyiapkan socket untuk menerima koneksi yang masuk dengan perintah *listen()*
   d. Menerima koneksi yang masuk ke server dengan perintah *accept()*
   e. Melakukan komunikasi (mengirim dan menerima data), dengan menggunakan perintah *write()* dan *read()*

Struktur Pengalamatan

Jaringan komputer dalam melakukan pengiriman data bisa diilustrasikan sebagai pengiriman surat. Supaya surat bisa terkirim secara benar maka alamat pengirim dan penerima harus tertulis dengan jelas dan lengkap.

Begitu juga dalam koneksi di socket, diperlukan *variable* yang dipakai untuk menyimpan *address client* dan *server*. *Variable* yang dipakai berupa *structure sockaddr dan sockaddr_in* pada include socket.h di direktory /usr/include/sys.

Address ini akan dipakai pada waktu melakukan connect(), bind() dan accept(). Dibawah ini adalah structute yang dipakai.

```
struct sockaddr {
unsigned short sa_family; /* protocol family */
char sa_data[14]; /* address data. */
};
```

```
struct sockaddr_in {
unsigned short sin_family; /* address family (always AF_INET)
*/
unsigned short sin_port; /* port num in network byte order */
struct in_addr sin_addr; /* IP addr in network byte order */
unsigned char sin_zero[8]; /* pad to sizeof(struct sockaddr) */
};
```

Contoh pemakaian struktur tersebut bisa dilihat pada tabel berikut ini:

```
struct sockaddr_in serveraddr;
/* fill in serveraddr with an address */
…
/* Connect takes (struct sockaddr *) as its second argument */
connect(clientfd, (struct sockaddr *) &serveraddr,
sizeof(serveraddr));
…
```

Langkah – Langkah Program di *Client*

1. Berikut ini adalah prosedur pembukaan koneksi client ke server pada *hostname:port* tertentu. Di dalamnya termasuk membuat socket, melakukan pengalamatan ke server dan melakukan koneksi ke server dengan perintah connect(). adalah sebagai berikut :

```
int open_clientfd(char *hostname, int port)
{
int clientfd;
struct hostent *hp;
struct sockaddr_in serveraddr;
```

```
if ((clientfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
return -1; /* check errno for cause of error */
/* Fill in the server's IP address and port */
if ((hp = gethostbyname(hostname)) == NULL)
return -2; /* check h_errno for cause of error */
bzero((char *) &serveraddr, sizeof(serveraddr));
serveraddr.sin_family = AF_INET;
bcopy((char *)hp->h_addr,
(char *)&serveraddr.sin_addr.s_addr, hp->h_length);
serveraddr.sin_port = htons(port);
/* Establish a connection with the server */
if (connect(clientfd, (struct sockaddr *) &serveraddr,
sizeof(serveraddr)) < 0)
return -1;
return clientfd;
}
```

Langkah demi langkah koneksi client ke server adalah sebagai berikut :

a. Membuat socket dengan perintah socket().

```
int clientfd; /* socket descriptor */
if ((clientfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
return -1; /* check errno for cause of error */
... (more)
```

`IF_INET` menunjukkan bahwa socket dihubungkan dengan protokol internet.
`SOCK_STREAM` menunjukkan bahwa program ini memakai *stream socket/TCP*, yang berari *connecton oriented.*

b. Selanjutnya setelah membuat socket melakukan pengalamatan ke server.

```
int clientfd; /* socket descriptor */
struct hostent *hp; /* DNS host entry */
struct sockaddr_in serveraddr; /* server’s IP address */
...
/* fill in the server's IP address and port */
if ((hp = gethostbyname(hostname)) == NULL)
return -2; /* check h_errno for cause of error */
bzero((char *) &serveraddr, sizeof(serveraddr));
serveraddr.sin_family = AF_INET;
bcopy((char *)hp->h_addr,
(char *)&serveraddr.sin_addr.s_addr, hp->h_length);
serveraddr.sin_port = htons(port);
```

c. Melakukan koneksi ke *server* dengan perintah *connect()*.

```
if (connect(sockfd, (struct sockaddr
*)&their_addr,sizeof(struct sockaddr)) == -1)
      {
            perror("connect");
            close(sockfd);
            exit(0);
      }
```

2. Melakukan komunikasi (mengirim dan menerima data), dengan menggunakan perintah *write()* dan *read()*

```
   if (write(sockfd, buff, strlen(buff))<0)
{
      close(sockfd);
      exit(1);
}
```

```
if (read(sockfd, buff, MAXBUFFER)<0)
   {
      printf("server : proses read dari socket error \n");
      close(sockfd);
      exit(1);
   }
```

3. Menutup hubungan dengan perintah *close()*;

Langkah – langkah Program di *Server*

1. Melakukan prosedur pembukaan koneksi yang di dalamnya berupa langkah – langkah : membuat socket, mengikat socket, menyiapkan socket menerima koneksi, pengalamatan socket.

```
int main(int argc, char **argv) {
int listenfd, connfd, port, clientlen;
struct sockaddr_in clientaddr;
struct hostent *hp;
char *haddrp;
port = atoi(argv[1]); /* the server listens on a port passed
on the command line */
listenfd = open_listenfd(port);
while (1) {
clientlen = sizeof(clientaddr);
connfd = Accept(listenfd, (SA *)&clientaddr, &clientlen);
hp = Gethostbyaddr((const char
*)&clientaddr.sin_addr.s_addr,
sizeof(clientaddr.sin_addr.s_addr), AF_INET);
haddrp = inet_ntoa(clientaddr.sin_addr);
printf("Fd %d connected to %s (%s:%s)\n",
connfd, hp->h_name, haddrp, ntohs(clientaddr.sin_port));
echo(connfd);
Close(connfd);
}
}
```

Langkah membuat socket, mengikat socket, menyiapkan socket menerima koneksi, pengalamatan socket berada pada prosedure *open_listenfd()*.

```
int open_listenfd(int port)
{
int listenfd, optval=1;
struct sockaddr_in serveraddr;
/* Create a socket descriptor */
if ((listenfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
return -1;
/* Eliminates "Address already in use" error from bind. */
if (setsockopt(listenfd, SOL_SOCKET, SO_REUSEADDR,
(const void *)&optval , sizeof(int)) < 0)
return -1;
/* Listenfd will be an endpoint for all requests to port
```

```
on any IP address for this host */
bzero((char *) &serveraddr, sizeof(serveraddr));
serveraddr.sin_family = AF_INET;
serveraddr.sin_addr.s_addr = htonl(INADDR_ANY);
serveraddr.sin_port = htons((unsigned short)port);
if (bind(listenfd, (SA *)&serveraddr, sizeof(serveraddr)) <
0)
return -1;
/* Make it a listening socket ready to accept
connection requests */
if (listen(listenfd, LISTENQ) < 0)
return -1;
return listenfd;
}
```

Langkah demi langkah membuat koneksi di server adalah sebagai berikut :

a. Membuat socket dengan perintah socket()

```
int listenfd; /* listening socket descriptor */
/* Create a socket descriptor */
if ((listenfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
return -1;
```

b. Melakukan pengalamatan.

```
struct sockaddr_in serveraddr; /* server's socket addr */
...
/* listenfd will be an endpoint for all requests to port
on any IP address for this host */
bzero((char *) &serveraddr, sizeof(serveraddr));
serveraddr.sin_family = AF_INET;
serveraddr.sin_addr.s_addr = htonl(INADDR_ANY);
serveraddr.sin_port = htons((unsigned short)port);
```

c. Mengikatkan socket kepada sebuah alamat network dengan perintah *bind()*

```
int listenfd; /* listening socket */
struct sockaddr_in serveraddr; /* server’s socket addr */
...
/* listenfd will be an endpoint for all requests to port
on any IP address for this host */
if (bind(listenfd, (struct sockaddr *)&serveraddr,
sizeof(serveraddr)) < 0)
return -1;
```

d. Menyiapkan socket untuk menerima koneksi yang masuk dengan perintah *listen()*

```
int listenfd; /* listening socket */
...
/* Make it a listening socket ready to accept connection
requests */
if (listen(listenfd, LISTENQ) < 0)
return -1;
return listenfd;
}
```

2. Looping utama adalah menerima koneksi, dan melakukan komunikasi data (mengirim dan menerima).

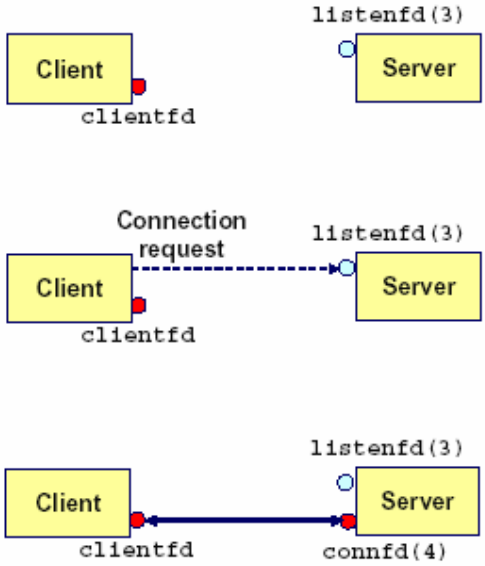
```
main() {
/* create and configure the listening socket */
while(1) {
/* Accept(): wait for a connection request */
/* echo(): read and echo input lines from client til EOF */
/* Close(): close the connection */
}
}
```

Langkah demi langkah looping utama adalah :

a. Menerima koneksi yang masuk ke server dengan perintah *accept()*

```
int listenfd; /* listening descriptor */
int connfd; /* connected descriptor */
struct sockaddr_in clientaddr;
int clientlen;
clientlen = sizeof(clientaddr);
connfd = Accept(listenfd, (SA *)&clientaddr, &clientlen);
```

Ilustrasi prosedur accept() jika digambarkan adalah sebagai berikut :



Gambar 6 Ilustrasi prosedur accept() antara client dan server

b. Melakukan komunikasi (mengirim dan menerima data), dengan menggunakan perintah *write()* dan *read()*

```
 if (write(sockfd, buff, strlen(buff))<0)
{
      close(sockfd);
      exit(1);
}
```

```
if (read(sockfd, buff, MAXBUFFER)<0)
   {
     printf("server : proses read dari socket error \n");
      close(sockfd);          exit(1);
   }
```

3. Menutup socket.

IV. Tugas Pendahuluan

1. Bagaimana konsep Client Server dalam jaringan komputer
2. Jelaskan secara singkat apa yang anda ketahui tentang Protokol Transport.
3. Dalam protokol transport terdapat dua protokol utama yaitu TCP and UDP, jelaskan perbedaan TCP dan UDP.
4. Berikan contoh aplikasi –aplikasi yang menggunakan protokol TCP, dan jelaskan bagaimana kerja aplikasi tersebut.
5. Jelaskan langkah dan prinsip kerja socket programming untuk komunikasi dua arah

V. Percobaan

1. Denga memakai editor vi tuliskan kembali program di bawah ini. Ada dua bagian program, *client* dan *server*. Simpan sesuai dengan nama yang ada pada *comment* program.
   `# vim /home/client.c`

```
/*
** client.c -- program client sederhana menggunakan stream socket
*/

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <netdb.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <sys/socket.h>

#define PORT 3333   // nomer port yang digunakan

#define MAXDATASIZE 100 // jumlah bytes maximal yang dikirimkan

int main(int argc, char *argv[])
{
        int sockfd, numbytes;
        char buf[MAXDATASIZE];
        struct hostent *he;
        struct sockaddr_in their_addr;      // informasi alamat server

        if (argc != 2) {
            fprintf(stderr,"usage: client hostname\n");
            exit(1);
        }
        if ((he=gethostbyname(argv[1])) == NULL) {  // mencari info tentang host
            perror("gethostbyname");
            exit(1);
        }
```

```
    if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
        perror("socket");
        exit(1);
    }

    their_addr.sin_family = AF_INET;            // host byte order
    their_addr.sin_port = htons(PORT);          // short, network byte order
    their_addr.sin_addr = *((struct in_addr *)he->h_addr);
    memset(&(their_addr.sin_zero), '\0', 8);    // lainnya diisi 0

    if (connect(sockfd, (struct sockaddr *)&their_addr,
                                sizeof(struct sockaddr)) == -1) {
        perror("connect");
        exit(1);
    }

    if ((numbytes=recv(sockfd, buf, MAXDATASIZE-1, 0)) == -1) {
        perror("recv");
        exit(1);
    }
    buf[numbytes] = '\0';
    printf("Received: %s",buf);
    close(sockfd);
    return 0;
}
```

Pada PC Server, ketikkan program berikut ini :

```
# vim /home/server.c
```

```
/*
 ** server.c -- program server sederhana menggunakan stream socket
*/

    #include <stdio.h>
    #include <stdlib.h>
    #include <unistd.h>
    #include <errno.h>
    #include <string.h>
    #include <sys/types.h>
    #include <sys/socket.h>
    #include <netinet/in.h>
    #include <arpa/inet.h>
    #include <sys/wait.h>
    #include <signal.h>

    #define MYPORT 3333    // nomer port yang digunakan

    #define BACKLOG 10     // jumlah koneksi yang diperbolehkan

    void sigchld_handler(int s)
    {
        while(wait(NULL) > 0);
    }
```

```
        int main(void)
    {
        int sockfd, new_fd;             // sock_fd ---> koneksi saat ini, new_fd ----> koneksi
baru
        struct sockaddr_in my_addr;    // ip address server
        struct sockaddr_in their_addr; // ip address client
        int sin_size;
        struct sigaction sa;
        int yes=1;

        if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
            perror("socket");
            exit(1);
        }

        if (setsockopt(sockfd,SOL_SOCKET,SO_REUSEADDR,&yes,sizeof(int)) == -1)
{
            perror("setsockopt");
            exit(1);
        }

        my_addr.sin_family = AF_INET;         // host byte order atau (big endian)
        my_addr.sin_port = htons(MYPORT);     // short, network byte order
```

```
        my_addr.sin_addr.s_addr = INADDR_ANY; // diisi dengan ip address server
        memset(&(my_addr.sin_zero), '\0', 8); // lainya diisi 0
        if (bind(sockfd, (struct sockaddr *)&my_addr, sizeof(struct sockaddr))
                                                    == -1) {
            perror("bind");
            exit(1);
        }

        if (listen(sockfd, BACKLOG) == -1) {
            perror("listen");
            exit(1);
        }

        sa.sa_handler = sigchld_handler;         // menghandle dead processes
        sigemptyset(&sa.sa_mask);
        sa.sa_flags = SA_RESTART;
        if (sigaction(SIGCHLD, &sa, NULL) == -1) {
            perror("sigaction");
            exit(1);
        }

        while(1) {  // main accept() loop
            sin_size = sizeof(struct sockaddr_in);
            if ((new_fd = accept(sockfd, (struct sockaddr *)&their_addr,
                  &sin_size)) == -1) {
                perror("accept");
                continue;
            }
            printf("server: got connection from %s\n",
                                        inet_ntoa(their_addr.sin_addr));
```

```
        if (!fork()) {          // jika child process
            close(sockfd);      // child process tidak perlu listener
            if (send(new_fd, "Pengiriman data dengan stream socket berhasil!\n", 50, 0)
                == -1)
                perror("send");
            close(new_fd);
            exit(0);
        }
        close(new_fd);          // menutup process
    }
    return 0;
}
```

2. Setelah selesai menulis dan menyimpan program, pastikan gcc sudah terinstall pada system operasi linux anda. Jalankan perintah :
   ```
   # dpkg –l | grep gcc
   ```
   Jika belum ter*install* lakukan *installasi* paket *gcc* beserta *library*nya.
   ```
   # apt-get install g++-3.4
   ```

3. Lakukan kompilasi program client dan server, dengan cara :
   ```
   # cd /home
   # gcc –o client client.c
   # gcc –o server server.c
   ```

4. Jalankan program server dan selanjutnya jalankan program client, dengan cara berikut ini :
   ```
   #./server
   #./client <no_ip_server>
   ```

5. Amati output yang dihasilkan.
6. Cek port yang digunakan oleh server
   ```
   # netstat –nlptu | grep server
   ```

7. Gunakan PC client lainnya, dan cobalah untuk mengakses ke server
   ```
   #./client <no_ip_server>
   ```

8. Ubahlah port pada client menjadi 5555, dan lakukan akses ke server lagi, amati apa yang terjadi.

   #define MYPORT 5555   // nomer port yang digunakan

9. Berikan komentar tiap baris pada program tersebut apa maksud dan kegunaan perintah diatas bila dihubungkan dengan stream socket (masukkan dalam laporan resmi).

VI. Laporan Resmi

Daftar Pertanyaan

1. Berikan kesimpulan praktikum yang anda lakukan.
2. Buatlah program memakai stream socket yang bisa mengirimkan inputan berupa character dari client dan diterima server kemudian ditampilkan di server apa yang dikirim client tersebut.