Pemrograman Socket dengan C

Ivan Irawan
ivanorma at indosat dot net dot id

Hmmm... Pas Socketnya! (1)

Abstraksi

Setiap kali kita berbicara masalah TCP/IP, maka akan selalu muncul istilah socket. Socket adalah mekanisme komunikasi yang memungkinkan terjadinya pertukaran data antar program atau proses baik dalam satu mesin maupun antar mesin. Pada setiap lingkungan sistem operasi yang mampu berkomunikasi dengan protokol TCP/IP, fasilitas socket selalu tersedia. Tulisan ini akan mencoba memberikan tutorial singkat mengenai dasar pemrograman socket dengan bahasa C. Lingkungan pemrograman yang digunakan dalam tutorial ini adalah Linux dan gcc, sehingga diperlukan sedikit penyesuaian jika akan digunakan pada lingkungan sistem operasi selain varian Unix.

Untuk dapat memahami tulisan ini, dibutuhkan pengetahuan dasar bahasa C, karena saya tidak akan menjelaskan hal dasar yang paling menjengkelkan bagi pemula seperti pointer. Tentu saja, seperti tulisan saya yang lain, Anda diwajibkan untuk gemar pada Star Trek, atau minimal Anda berpura-pura gemar Star Trek. Jika Anda tidak dapat melakukannya, saya tidak menjamin jika Anda akan mendapatkan 'Socket' yang pas.

Hubungkan Saya Dengan Kapten Piccard!

Sebuah panggilan transmisi *subspace* masuk ke USS Enterprise D. Panggilan tersebut diterima oleh Data sebagai *Communication Officer*. Data membuka komunikasi, ternyata panggilan tersebut dari Admiral Starfleet untuk Kapten Piccard. Data mentransfer panggilan tersebut ke kamar Kapten Piccard. Komunikasi terjadi setelah Kapten Piccard mengangkat perangkat komunikator dan berbicara dengan Admiral Starfleet.

Pada saat yang sama Data kembali menerima panggilan, dan setelah komunikasi dibuka, ternyata panggilan yang masuk adalah untuk *First Officer* Riker. Seperti sebelumnya Data mengarahkan panggilan ke tempat yang tepat sehingga komunikasi dapat terjadi. Demikian seterusnya.

Hal yang serupa terjadi pada koneksi pada socket, setiap permintaan sambungan oleh client akan di routing menuju tujuan yang tepat. Sedangkan di sisi server, aplikasi atau program terlebih dahulu harus membuat socket yang dapat diakses oleh client yang melakukan request ke server. Socket semacam ini dikenal sebagai socket server yang memiliki fungsi serupa dengan tugas Data pada analogi di atas. Agar socket server dapat tetap melayani permintaan sambungan yang masuk, maka koneksi yang terjadi akan dialihkan ke socket lain yang dibentuk oleh aplikasi server. Ini serupa dengan proses transfer panggilan komunikasi ke orang yang tepat.

Secuil Cerita Socket

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 variant Unix dan Linux mengadopsi BSD Socket.

Pada lingkungan Unix, socket memberikan keleluasaan pemrograman gaya Unix yang terkenal dengan ideologinya,

> *Semua di Unix/Linux adalah file*.

Komunikasi antar program dapat berlangsung lewat penggunaan deskriptor file standar Unix dengan bantuan socket. Bagi Anda yang belum kenal tentang deskriptor file Unix, secara garis besarnya adalah seperti berikut ini.

Semua entitas yang ada pada Unix atau Linux selalu didefinisikan sebagai file. Perangkat-perangkat dalam komputer, seperti hardisk, mouse, bahkan perangkat I/O seperti Serial Port, Paralel Port, USB, selalu didefinisikan sebagai file dalam sistem file Unix. Contohnya untuk port serial mungkin akan didefinisikan sebagai file `/dev/ttyS1`. Secara konsep, untuk dapat mengirimkan dan menerima data lewat port serial, Anda harus membuka file `/dev/ttyS1` dengan sebuah fungsi misalkan `open()`

yang akan menghasilkan sebuah nilai integer. Nilai integer ini akan disimpan pada sebuah variable tertentu yang dikenal dengan deskriptor file. Pengiriman dan penerimaan data dilakukan dengan bantuan deskriptor file ini sebagai referensi tempat data tersebut dikirim dan diterima. Anda dapat membuka referensi mengenai ini pada referensi Unix/Linux.

Pada lingkungan sistem operasi MS Windows, kita kenal Winsock (Windows Socket), yang walaupun gaya pemrograman winsock ini sedikit berbeda dengan socket di Unix. Meskipun demikian, program yang dihasilkan oleh winsock dipastikan dapat berkomunikasi dengan baik dengan program yang dihasilkan oleh socket BSD, sepanjang keduanya mengadopsi protokol yang sama.

Komunikasi antar proses dengan menggunakan socket ini merupakan pengembangan lebih lanjut dari "teknologi pipes" di dalam lingkungan Unix. Socket ini dapat digunakan seperti jika penggunaan pipes di unix. Keunggulan dari penggunaan socket ini dibanding apabila menggunakan pipes biasa adalah anda dapat melakukan komunikasi antar proses/program melalui jaringan berbasis yang TCP/IP tentunya, bahkan dengan program lain yang berjalan pada platform non-unix seperti Microsoft Windows, sepanjang program tersebut berbicara dalam protokol transfer yang sama.

Fasilitas-fasilitas yang disediakan oleh mesin unix seperti rlogin, ssh, ftp, dan lain-lain menggunakan socket sebagai sarana komunikasi mereka. Socket dibentuk dan digunakan dengan cara yang berbeda dengan proses pipes di unix.

Komunikasi socket terutama diciptakan untuk tujuan menjembatani komunikasi antara dua buah program yang dijalankan pada mesin yang berbeda. Jangan khawatir, ini tentu saja berarti dua program pada mesin yang sama dapat juga saling berkomunikasi. Kelebihan lain dari komunikasi socket adalah mampu menangani banyak klien sekaligus (multiple clients).

Jangan-jangan Samakan Dia Dengan Yang Lain

Ada dua golongan socket di Unix yang paling umum dipakai yaitu:

1. Socket Lokal atau `AF_UNIX`
2. Socket Networking atau `AF_INET`

Socket Lokal adalah socket yang melakukan komunikasi dengan perantaraan sebuah file yang biasanya diletakkan pada direktori `/tmp` atau `/usr/tmp` ataupun `/var/tmp`. Socket semacam ini digunakan umumnya terbatas untuk komunikasi antar aplikasi dalam satu mesin.

Socket Networking ditujukan untuk komunikasi antar aplikasi antar mesin dalam lingkungan jaringan TCP/IP. Identifikasi socket dilakukan dengan sebuah *service identifier* yaitu berupa nomor port TCP/IP yang dapat di sambung oleh client. Kita akan membahas socket networking pada artikel ini.

Golongan socket lainnya masih banyak misalnya `AF_OSI`, `AF_NS`, namun kita tidak membahasnya kali ini.

Socket Networking memiliki beberapa jenis, yang paling umum digunakan yaitu:

3. Socket Stream atau `SOCK_STREAM`

4. Socket Datagram atau `SOCK_DGRAM`

Socket Stream adalah socket komunikasi *full-duplex* berbasis aliran (*stream*) data. Pada model komunikasi Socket Stream, koneksi dua aplikasi harus dalam kondisi tersambung dengan benar untuk dapat bertukar data. Ini dapat dianalogikan seperti komunikasi telepon. Jika sambungan telepon di salah satu titik putus, maka komunikasi tidak dapat terjadi. Koneksi model seperti ini akan menjamin data dapat dipertukarkan dengan baik, namun memiliki kelemahan dalam hal penggunaan jalur data yang relatif besar dan tidak boleh terputus.

Socket Datagram berkomunikasi dengan cara yang berbeda. Socket ini tidak membutuhkan koneksi yang tersambung dengan benar untuk mengirimkan dan menerima data. Model koneksi semacam ini tidak dapat menjamin data dapat dipertukarkan dengan baik, namun memiliki keunggulan dalam hal penggunaan jalur data yang minimal. Socket Datagram dapat dianalogikan dengan komunikasi yang terjadi pada kelas, misalnya pada saat guru melakukan *broadcasting* materi pelajaran untuk diterima oleh setiap murid. Tidak ada yang dapat menjamin materi pelajaran dapat diterima oleh semua murid dengan baik, kecuali diterapkan metoda *rechecking*. Rechecking ini dapat dilakukan baik oleh guru maupun murid. Guru bertanya untuk memastikan jawaban dari murid benar, atau murid bertanya untuk memastikan kebenaran materi yang diterimanya. Socket Datagram pun menggunakan metoda ini untuk menjamin pengiriman data dapat dilakukan dengan baik.

Cukup! Kita mulai pusing dengan teori, mari kita masuk ke praktek. Siapkan sistem Unix/Linux Anda, Compiler gcc, dan file header C yang dibutuhkan. Anda dapat menggunakan text editor yang paling Anda sukai untuk mengetik kode sumber program. Siapkan kopi, dan jangan lupa tablet *analgetic antipiretic* favorit Anda kalau-kalau efek samping membaca artikel ini muncul.

USS Enterprise Di Sini...

Kita akan mencoba untuk membuat protokol identifikasi kapal *starfleet* yang akan dipasang pada USS Enterprise. Protokol ini memiliki rancangan yang bersifat RSS artinya Rancangan Sangat Sederhana yaitu:

- Server Identifikasi akan bersiap menerima koneksi di port 26971, alasannya sangat sederhana juga, karena saya lahir pada tanggal tersebut. Barang siapa yang telah membaca artikel ini, mohon jangan lupa mengucapkan selama ulang tahun pada tanggal tersebut. Ini bukan permintaan tapi paksaan.

- Server akan menerima koneksi, mengirim data identifikasi ke koneksi yang masuk dan langsung memutus koneksi tersebut.

Saat ini kita coba untuk membuat servernya dulu. Socket yang digunakan adalah socket `AF_INET` dengan jenis `SOCK_STREAM`. Socket didefinisikan dengan empat buah komponen yaitu:

1. Identifikasi Nomor atau Alamat Host Remote
2. Nomor Port Host Remote
3. Identifikasi Nomor atau Alamat Host Lokal
4. Nomor Port Host Lokal

Untuk server kita, yang penting didefinisikan di awal adalah poin 3 dan poin 4. Poin 1 dan poin 2 akan kita dapatkan saat ada client yang masuk menghubungi server.

Ada lima langkah dasar untuk pemrograman socket server adalah:

1. Membuat socket dengan perintah `socket()`
2. Mengikatkan socket kepada sebuah alamat network dengan perintah `bind()`
3. Menyiapkan socket untuk menerima koneksi yang masuk dengan perintah `listen()`
4. Menerima koneksi yang masuk ke server dengan perintah `accept()`
5. Melakukan komunikasi (mengirim dan menerima data), dengan menggunakan perintah `write()` dan `read()`

Bagaimana implementasinya? Coba ketik kode sumber berikut ini dan simpan dengan nama `server1.c`

```
/* Program Server 1 : Identifikasi USS Enterprise */

#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>



/* Fungsi Untuk Menangani Error Yang Muncul */

void error(char *msg)
{
        perror(msg);
        exit(1);
}

/* Fungsi Utama */

int main(int argc, char *argv[])
{
        int sockdf, sockdfbaru, noport, pjclient, n;
        struct sockaddr_in alamat_srv, alamat_cli;
        char buffer[256];

        if (argc < 2)
        {
                fprintf(stderr,
```

```
                "ERROR, Anda belum mendefinisikan Nomor Port!\n");
                exit(1);
        }

        noport = atoi(argv[1]);

        sockdf = socket(AF_INET, SOCK_STREAM, 0);
        if (sockdf < 0)
                error ("ERROR Inisiasi Socket");

        bzero((char *) &alamat_srv, sizeof(alamat_srv));

        alamat_srv.sin_family = AF_INET;
        alamat_srv.sin_port = htons(noport);
           alamat_srv.sin_addr.s_addr = INADDR_ANY;

        if (bind(sockdf, (struct sockaddr *) &alamat_srv,
                sizeof(alamat_srv)) < 0 )
                        error("Error mengikatkan socket ke Alamat Server\n");

        listen(sockdf,5);

        pjclient = sizeof(alamat_cli);
        sockdfbaru = accept(sockdf, (struct sockaddr *) &alamat_cli,
&pjclient);
        if (sockdfbaru < 0)
                error("ERROR pada saat menerima koneksi\n");

        bzero(buffer,256);
        n = read(sockdfbaru, buffer, 255);
        if (n < 0) error("ERROR penerimaan data");

        printf("Ada Permintaan Identifikasi Kapal dari %s \n",
                                        inet_ntoa(alamat_cli.sin_addr));
        printf("Pesan : %s \n",buffer);


        n = write(sockdfbaru, "Ini adalah USS Enterprise D.",28);
        if (n < 0) error("ERROR pengiriman data");

        return 0;
}
```

Mari kita satu-persatu mengartikan isi kode sumber di atas, pertama kali kita harus mendefinisikan file header yang kita gunakan yaitu:

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
```

Header `stdio.h` diperlukan untuk fungsi-fungsi standar input/output yang biasa digunakan di hampir semua program C. Header file `sys/types.h` berisi definisi dari beberapa jenis data yang dipergunakan pada pemanggilan sistem (*system call*). Jenis data tersebut digunakan pada dua header berikutnya. Header `sys/socket.h` berisi sejumlah struktur yang digunakan pada pemrograman socket, sementara header `netinet/in.h` berisi struktur dan konstanta yang dipergunakan untuk alamat domain internet.

```
void error(char *msg)
{
        perror(msg);
        exit(1);
}
```

Fungsi di atas digunakan untuk menangani kesalahan yang terjadi. Jika terjadi kesalahan maka tampilkan pesan error kemudian program terhenti.

Berikutnya adalah fungsi utama, yang akan menangkap argumen lain selain nama program yaitu:

```
int main(int argc, char *argv[])
{
  .....
}
```

Argumen `argc` memberikan jumlah argumen pada saat pemanggilan program, sementara array `argv[]` memberikan nilai dari setiap argumen pada saat pemanggilan program. Sebelum saya lupa, semua ini saya lakukan sebagai respon akibat beberapa keberatan yang muncul terhadap tindakan saya yang otoriter menetapkan port 26971 sebagai port program, maka saya sedikit memodifikasi kode sumber program server ini, sehingga setiap pengguna program dapat menentukan sendiri port yang diinginkan dengan cara pemanggilan program dilengkapi dengan argumen port. Bukankah saya amat bijaksana dan penuh pengertian?

Kode sumber berikutnya berada dalam blok fungsi `main()`.

```
int sockdf, sockdfbaru, noport, pjclient, n;
struct sockaddr_in alamat_srv, alamat_cli;
char buffer[256];
```

Kali ini kita mendeklarasikan beberapa variabel yang akan kita gunakan dalam program. Variabel `sockdf` adalah variabel untuk menyimpan nilai deskriptor file socket server, dan variabel `sockdfbaru` digunakan untuk menyimpan nilai deskriptor file socket yang digunakan untuk melayani koneksi yang masuk. Kedua variabel tersebut akan dihasilkan dari pemanggilan sistem socket, dan digunakan sebagai referensi pada saat koneksi dan komunikasi data. Port dari server akan disimpan pada variabel `noport` dan panjang dari alamat client akan disimpan pada variabel `pjclient`. Untuk keperluan pengecekan kesalahan pada proses pengiriman dan penerimaan data digunakan variabel `n` untuk menyimpan hasil.

Variabel `alamat_srv` digunakan untuk menyimpan alamat server dan variabel `alamat_cli` digunakan untuk menyimpan alamat client. Kedua variable ini berjenis `struct sockaddr_in` yang didefinisikan pada file `netinet/in.h`. Berikut ini adalah strukturnya:

```
struct sockaddr_in
{
        short          sin_family;    /* harus berjenis AF_INET */
        u_short                sin_port;
        struct         in_addr sin_addr;
        char           sin_zero[8];   /* Tidak digunakan harus bernilai 0 */
};
```

Sementara `struct in_addr` didefinisikan dalam file header yang sama dan hanya memiliki satu field, sebuah variabel jenis long unsigned dinamakan `s_addr`.

Variabel `buffer` digunakan untuk menyimpan data yang diterima dari socket.

```
if (argc < 2)
{
        fprintf(stderr,
        "ERROR, Anda belum mendefinisikan Nomor Port!\n");
        exit(1);
}
```

Fungsi di atas ini digunakan untuk melakukan pengecekan jumlah argumen pemanggilan program. Argumen minimal harus dua, argumen pertama adalah nama program yang dipanggil dan argumen kedua adalah nomor port yang digunakan server. Jika kurang dari dua, maka program akan terhenti. Agar anda tidak bingung, jika anda memanggil program dari shell:

```
$ ./server1
```

Sesungguhnya anda telah memberikan sebuah argumen, yaitu nama program server1. Untuk menjalankan program ini, Anda harus menambahkan satu lagi argumen , yaitu port dengan cara:

```
$ ./server1 26971
```

Angka 26971 adalah argumen kedua yang akan dibaca oleh program sebagai nomor port pada kode berikut ini.

```
noport = atoi(argv[1]);
```

Fungsi `atoi()` adalah fungsi untuk mengubah ascii (a) menjadi (to) integer, sehingga disingkat dengan `atoi()`. Variabel `argv[1]` adalah isi dari argumen kedua pada pemanggilan program.

```
sockdf = socket(AF_INET, SOCK_STREAM, 0);
if (sockdf < 0)
        error ("ERROR Inisiasi Socket");
```

Kode di atas adalah untuk membuka socket golongan `AF_INET` dengan jenis `SOCK_STREAM`. Pembukaan socket ini akan menghasilkan nilai deskriptor file yang disimpan dakam variabel `sockdf`. Deskriptor file akan bernilai -1 jika terjadi kesalahan pada saat pembukaan socket. Pada kondisi ini maka program akan memberikan pesan kesalahan.

```
bzero((char *) &alamat_srv, sizeof(alamat_srv));
```

Fungsi `bzero()` akan mengeset seluruh nilai pada suatu buffer menjadi zero. Argumen pertama dari fungsi ini adalah penunjuk (*pointer*) dari buffer dan argumen kedua adalah ukuran dari buffer tersebut. Baris kode di atas akan menginisiasi variabel `alamat_srv` menjadi zero.

```
alamat_srv.sin_family = AF_INET;
alamat_srv.sin_port = htons(noport);
        alamat_srv.sin_addr.s_addr = INADDR_ANY;
```

Variabel `alamat_srv` memiliki beberapa field. Field `sin_family` harus diset menjadi konstanta simbolic `AF_INET`. Field `sin_port` harus diisikan dengan nomor port. Agar portabilitas program terjamin di semua lingkungan pemrograman, maka digunakan fungsi `htons()` untuk mengubah format biner komputer host ke standar network dalam jenis bilangan short. Fungsi `htons()` berasal dari host (h), menjadi (to), network (n), short (s), sehingga disingkat menjadi `htons()`. Detail keterangan mengapa kita harus melakukan konversi ini akan dijelaskan kemudian. Kata kuncinya adalah portabilitas program.

Field `sin_addr` berisi satu field long unsigned `s_addr`, yang diisi dengan alamat IP dari host. Untuk kode server, berarti adalah alamat IP dari mesin di mana server berjalan, yang dapat digantikan dengan konstanta simbolic `INADDR_ANY`.

```
if (bind(sockdf, (struct sockaddr *) &alamat_srv,
        sizeof(alamat_srv)) < 0 )
                error("Error mengikatkan socket ke Alamat Server\n");
```

Kode di atas digunakan untuk mengikatkan socket yang telah dibuka yang ditunjukkan dalam deskriptor file `sockdf`, ke alamat IP dan nomor port host yang ditunjukkan oleh variabel `alamat_srv`. Baris kode di atas juga sekaligus melakukan pengecekan error jika terjadi kesalahan pada saat pengikatan socket. Fungsi `bind()`

membutuhkan argumen kedua berupa `struct sockaddr`, padahal `alamat_srv` berjenis `sockaddr_in`, sehingga perlu di*casting* dengan jenis yang tepat.

```
listen(sockdf,5);
```

Kali ini kita akan membuat socket server yang telah kita siapkan untuk menunggu koneksi yang datang. Kita menggunakan fungsi `listen()`. Argumen kedua adalah antrian backlog, yakni jumlah koneksi yang dapat mengantri saat proses sedang menangani suatu koneksi. Nilai 5 adalah maksimum backlog yang umum diizinkan pada kebanyakan sistem.

```
pjclient = sizeof(alamat_cli);
sockdfbaru = accept(sockdf, (struct sockaddr *) &alamat_cli,
&pjclient);
if (sockdfbaru < 0)
        error("ERROR pada saat menerima koneksi\n");
```

Fungsi accept() akan memblok proses sampai sebuah koneksi client terhubung dengan server. Kejadian ini akan membangunkan proses dari tidurnya . Fungsi accept() akan mengembalikan nilai berupa deskriptor file baru. Seluruh komunikasi pada koneksi ini harus dilakukan dengan deskriptor file baru ini. Argumen kedua adalah penunjuk (*pointer*) referensi dari alamat client pada ujung lain dari koneksi, dan argumen ketiga adalah ukuran dari struktur data.

```
bzero(buffer,256);
n = read(sockdfbaru, buffer, 255);
if (n < 0) error("ERROR penerimaan data");

printf("Ada Permintaan Identifikasi Kapal dari %s \n",
                                        inet_ntoa(alamat_cli.sin_addr));
printf("Pesan : %s \n",buffer);
```

Kode di atas hanya akan beroperasi jika suatu client terhubung dengan server. Kode sumber menginisiasi variabel `buffer` dengan fungsi `bzero()`. Variabel `buffer` ini akan digunakan untuk membaca data dari socket dengan fungsi `read()`. Pembacaan ini akan mengambil data dari socket menggunakan deskriptor file baru yang dihasilkan oleh `accept()` bukannya menggunakan deskriptor file yang asli dihasilkan dari fungsi `socket()`. Fungsi `read()` akan memblok proses hingga muncul sesuatu untuk dibaca pada socket, misalnya ketika client telah mengekskusi fungsi `write()`. Jumlah karakter yang dibaca pada socket adalah sejumlah karakter yang ada pada socket atau 255 tergantung mana yang lebih kecil. Ini ditentukan pada argument ketiga dari fungsi `read()`. Fungsi `read()` menghasilkan jumlah karakter yang dibaca yang disimpan sementara dalam variabel `n`. Data hasil pembacaan yang disimpan dalam variabel buffer akan ditampilkan pada standar I/O demgam perintah `printf()`.

```
n = write(sockdfbaru, "Ini adalah USS Enterprise D.",28);
if (n < 0) error("ERROR pengiriman data");
```

Langkah selanjutnya adalah mengirim data identifikasi kapal kepada client melalui socket dengan menggunakan fungsi `write()`. Argumen terakhir dalam fungsi `write()` menyatakan jumlah karakter yang dikirim ke client melalui socket. Pesan ini yang nantinya akan diterima client sebagai identifikasi kapal *starfleet*.

```
return 0;
```

Kode terakhir pada program server adalah mengembalikan nilai 0 untuk fungsi `main()` sekaligus menghentikan jalannya program.

Jika Anda telah selesai mengetik kode program `server1.c` maka kini giliran kita untuk melakukan kompilasi, sebagai berikut:

```
$ gcc -o server1 server1.c
```

Jika tidak ditemukan kesalahan, maka segera akan terbentuk file `server1`. Kita harus memeriksa status file `server1`, jika belum memiliki pemission executable, maka lakukan:

```
$ chmod a+x server1
```

Jalankan server dengan perintah:

```
$ ./server1 26971
```

Jika tidak muncul kesalahan, maka program `server1` telah siap menerima koneksi dari client pada port 26971. Apakah ini sudah cukup? Tentu saja belum. Kini kita perlu untuk membuat program client yang akan menghubungi server.

Halo, Kaukah Di Sana?

Untuk membuat program client yang bertugas menghubungi server, maka langkah-langkahnya adalah sebagai berikut:

1. Membuat socket dengan perintah `socket()`
2. Menghubungi server dengan `connect()`
3. Melakukan komunikasi (mengirim dan menerima data), dengan menggunakan perintah `write()` dan `read()`

Ketik kode di bawah ini dan simpan dengan nama `client1.c`.

```
/* Program Client : Permintaan Identifikasi */
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

void error(char *msg)
{
       perror(msg);
       exit(0);
}

int main(int argc, char *argv[])
{
       int sockdf, noport, n;
       struct sockaddr_in alamat_srv;
       struct hostent *server;
       char buffer[256];



       if (argc < 3)
       {
              fprintf(stderr, "Gunakan port nama host %s\n", argv[0]);
              exit(0);
       }

       noport = atoi(argv[2]);
       sockdf = socket(AF_INET, SOCK_STREAM, 0);
       if (sockdf < 0)
              error("ERROR membuka socket");
```

```
        server = gethostbyname(argv[1]);
        if (server == NULL)
        {
               fprintf(stderr, "ERROR, host tidak ditemukan\n");
               exit(0);
        }

        bzero((char *) &alamat_srv, sizeof(alamat_srv));
        alamat_srv.sin_family = AF_INET;
        bcopy( (char *)server->h_addr,
                     (char *)&alamat_srv.sin_addr.s_addr,
                     server->h_length);
        alamat_srv.sin_port = htons(noport);

        if (connect(sockdf, (struct sockaddr *)&alamat_srv,
               sizeof(alamat_srv)) < 0)
               error("ERROR menghubungi server");
          n = write(sockdf,"Meminta Identifikasi Kapal",26);
        if (n < 0)
               error("ERROR Mengirim data ke Socket");
        bzero(buffer,256);
        n = read(sockdf,buffer,255);
        if (n < 0)
               error("ERROR Membaca data dari Socket");
        printf("Kapal Mengirim Identifikasi\n");
        printf("Pesan Diterima : %s \n", buffer);
        return 0;
}
```

Mari kita kupas baris demi baris program Client ini.

```
/* Program Client : Permintaan Identifikasi */
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

void error(char *msg)
{
       perror(msg);
       exit(0);
}
```

Tidak ada yang berbeda pada baris kode ini kecual tambahan file header `netdb.h` yang diperlukan untuk mendefinisikan struktur `struct hostent` yang akan kita bahas kemudian. Fungsi penanganan kesalahan juga sama.

```
int main(int argc, char *argv[])
{
       ....
}
```

Fungsi utama tidak berbeda dengan program server, hanya pada program client kita akan menggunakan 2 argumen tambahan selain nama program yaitu nama host dan port dari host. Baris kode berikutnya adalah baris kode di dalam blok fungsi `main()`.

```
        int sockdf, noport, n;
        struct sockaddr_in alamat_srv;
        struct hostent *server;
        char buffer[256];
```

Pada deklarasi variabel kita akan mendefinisikan `sockdf` untuk menyimpan deskriptor file socket, `noport` untuk menyimpan port dari server, `n` untuk keperluan pengecekan kesalahan pada pengiriman dan penerimaan data. Kita tidak perlu mendeklarasikan

variabel untuk alamat client karena tidak dibutuhkan, jadi cukup deklarasikan variabel `alamat_srv` untuk menyimpan data mengenai host/server yang dihubungi dengan jenis `struct sockaddr_in`. Variabel `buffer` juga kita deklarasikan untuk menyimpan hasil penerimaan data dari socket. Hal yang baru adalah deklarasi variabel `server` yang merupakan pointer dengan jenis `struct hostent`. Struktur hostent sendiri memiliki definisi sesuai dengan `netdb.h`:

```
struct hostent
{
        char    *h_name;            /* nama resmi dari host */
        char    **h_aliases;        /* daftar alias */
        int     h_addrtype;         /* jenis alamat host */
        int     h_length;           /* panjang alamat */
        char    **h_addr_list;      /* daftar alamat dari name server */
#define h_addr h_addr_list[0]     /* alamat, kompatibilitas balik */
};
```

Ini akan mendefinisikan suatu host dalam internet. Anggota struktur ini adalah:

- Nama resmi dari host (`h_name`)
- Nama alternatif dari host (`h_aliases`)
- Jenis alamat host, selalu jenis `AF_INET` (`h_addrtype`)
- Ukuran alamat dalam byte (`h_length`)
- Pointer daftar alamat jaringan untuk host. Alamat host diberikan dalam urutan byte jaringan (*network byte order*)

Field `h_addr` adalah alias untuk alamat pertama dari array alamat jaringan.

```
if (argc < 3)
{
        fprintf(stderr, "Gunakan port nama host %s\n", argv[0]);
        exit(0);
}
```

Kode program di atas adalah untuk melakukan pemeriksaan kelengkapan dari argumen pemanggilan program. Jika argumen tidak lengkap (kurang dari 3) maka program dihentikan setelah sebelumnya memberikan keterangan kesalahannya.

```
noport = atoi(argv[2]);
sockdf = socket(AF_INET, SOCK_STREAM, 0);
if (sockdf < 0)
        error("ERROR membuka socket");
```

Kita sekarang akan mengisi variabel `noport` dengan argumen ketiga (indeks 2 karena argumen pertama berindeks 0) yang didapat dari pemanggilan program. Setelah itu socket dibuka dengan parameter `AF_INET` dan `SOCK_STREAM`.

```
server = gethostbyname(argv[1]);
if (server == NULL)
{
        fprintf(stderr, "ERROR, host tidak ditemukan\n");
        exit(0);
}
```

Baris di atas akan mencari alamat host berdasarkan nama yang diberikan pada argumen kedua dan menyimpannya pada variabel server. Fungsi yang digunakan adalah `gethostbyname()`. Pointer dari variabel itu akan digunakan pada struct hostent. Field `char *h_addr` akan berisi alamat IP. Jika struktur ini `NULL`, berarti sistem tidak mampu menentukan alamat suatu host berdasarkan namanya.

```
bzero((char *) &alamat_srv, sizeof(alamat_srv));
alamat_srv.sin_family = AF_INET;
bcopy( (char *)server->h_addr,
             (char *)&alamat_srv.sin_addr.s_addr,
             server->h_length);
alamat_srv.sin_port = htons(noport);
```

Kode di atas adalah kode untuk mengeset field-field dalam variabel `alamat_srv`. Tidak jauh berbeda dengan kode yang ada di program server kecuali pada pengisian field `sin_addr.s_addr`. Karena field `server->h_addr` adalah karakter string, maka digunakan fungsi `bcopy()`, yang akan mengisi `alamat_srv.sin_addr.s_addr` dengan isi variabel `server->h_addr` dalam ukuran byte `server->h_length`.

```
if (connect(sockdf, (struct sockaddr *)&alamat_srv,
       sizeof(alamat_srv)) < 0)
       error("ERROR menghubungi server");
```

Kini kita akan membuat koneksi ke server sekaligus mengecek hasil koneksinya. Fungsi yang digunakan adalah `connect()` yang akan memberikan hasil -1 jika gagal atau 0 jika berhasil. Argumen yang dibutuhkan adalah deskriptor file dari socket (`sockdf`), alamat server dalam jenis `struct sockaddr`, dan ukuran dari alamat server.

```
 n = write(sockdf,"Meminta Identifikasi Kapal",26);
if (n < 0)
       error("ERROR Mengirim data ke Socket");
```

Kode berikutnya adalah mengirim pesan ke server dan melakukan pengecekan kesalahan jika terjadi.

```
bzero(buffer,256);
n = read(sockdf,buffer,255);
if (n < 0)
       error("ERROR Membaca data dari Socket");
```

Setelah mengirim pesan ke server, maka client akan membaca pesan identifikasi kapal dari server.

```
printf("Kapal Mengirim Identifikasi\n");
printf("Pesan Diterima : %s \n", buffer);
```

Kemudian akan menampilkan ke layar hasil penerimaan identifikasi kapal.

```
return 0;
```

Program dihentikan dengan memberikan nilai 0 terhadap fungsi main().

Selesai mengetik kode program `client1.c` maka kini giliran kita untuk melakukan kompilasi, sebagai berikut:

```
$ gcc -o client1 client1.c
```

Jika tidak ditemukan kesalahan, maka segera akan terbentuk file `client1`. Kita harus memeriksa status file `client1`, jika belum memiliki pemission executable, maka lakukan:

```
$ chmod a+x client1
```

Jalankan server dengan perintah:

```
$ ./client1 localhost 26971
```

Maka pada program client akan muncul tampilan:

```
Kapal Mengirim Identifikasi
Pesan Diterima : Ini adalah USS Enterprise D.
```

Tentu saja tampilan ini akan muncul jika program server juga dijalankan. Sebaiknya program server dan program client dijalankan dalam dua konsol yang terpisah atau dari mesin yang berbeda. Kemudian program client akan berhenti dan terminal kembali ke shell. Jika kita melongok ke konsol tempat program server dijalankan, maka akan muncul tampilan:

```
Ada Permintaan Identifikasi dari 127.0.0.1
Pesan : Meminta Identifikasi Kapal
```

Kemudian program server juga akan terhenti dan terminal kembali ke shell. Hal ini berarti Anda telah berhasil membuat program socket sederhana untuk USS Enterprise dan untuk setiap kapal yang meminta identifikasi kapal.

Mungkin Anda belum puas, karena setiap selesai melakukan menjawab identifikasi, program server selalu terhenti, dan Anda harus menjalankannya kembali secara manual. Sangat merepotkan operator program. Adakah caranya agar semua ini berlangsung dengan otomatis? Artinya program server pada USS Enterprise akan selalu siap menerima permintaan identifikasi kapal dari dari mana pun, kapan pun, walaupun sedang melayani permintaan identifikasi kapal tertentu. Program juga tidak akan terhenti setelah melayani permintaan identifikasi.

Model seperti ini dikenal dengan koneksi multi client. Kita akan membahasnya pada bagian kedua dari artikel ini. Sebagai tambahan sebelum melangkah lebih lanjut kita akan membahas sedikit mengenai urutan byte jaringan (*network byte order*).

Aku Begini, Engkau Begitu, Sama Saja

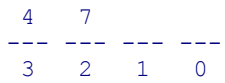
Pada program yang telah kita buat, telah kita kenal fungsi htons(). Sesungguhnya apa perlunya kita menggunakan fungsi ini? Mari kita dengar dongeng ini.

Berkembangnya beragam jenis mesin pada dunia komputasi ini ternyata juga menyebabkan variasi dalam pengolahan data digital di setiap jenis mesin. Ada mesin-mesin yang mengolah data dengan cara big endian ada pula yang little endian. Sementara hubungan antar mesin telah menggunakan cara pengolahan yang standar. Karena bahasa C berarti pula portabilitas dari satu mesin ke mesin yang lain, agar program kita dapat dikompilasi dan dijalankan pada mesin-mesin yang berbeda, maka penting sekali menghargai perbedaan ini.

Berikut adalah contohnya, mesin big endian, contoh mesin SUN, akan menyimpan data 4 + 7 * 256 dalam bentuk:

```
          7   4
--- --- --- ---
 3   2   1   0
```

Sementara untuk angka yang sama, mesin little endian seperti x86 akan menyimpan dengan cara:



Bayangkan jika mesin SUN mengirim dara 4 + 7 * 256 tanpa dikonversi ke mesin x86. Mesin x86 akan membaca data menjadi 4 * 16777216 + 7 * 65536.

Fungsi konversi byte baik dari host ke network dan dari network ke host akan sangat membantu menjamin portabilitas. Pada mesin-mesin yang menggunakan pengolahan byte serupa dengan network byte order, fungsi konversi tidak akan melakukan apa pun, namun pada mesin-mesin yang menggunakan pengolahan byte berbeda dengan network byte order, fungsi ini akan menyesuaiakan byte data sehingga tidak terjadi kesalahan pembacaan antara host (mesin) maupun jaringan (network). Sebagai pemrogram kita cukup memastikan bahwa kita menggunakan fungsi ini, maka compiler C akan melakukan tugasnya dengan baik tanpa kita perlu lagi memodifikasi program.

Pada dasarnya kita dapat mengkonversi dua jenis bilangan yaitu short (2 byte) dan long (4 byte). Sementara konversi dilakukan dari host (h) ke network (n) atau sebaliknya, sehingga kita kenal fungsi konversi urutan byte jaringan yaitu:

- `htons()` konversi urutan byte dari host ke network jenis short
- `htonl()` konversi urutan byte dari host ke network jenis long
- `ntohs()` konversi urutan byte dari network ke host jenis short
- `ntohl()` konversi urutan byte dari network ke host jenis long

Sebagai programmer, yakinkan bahwa setiap penggunaan data dari host ke jaringan telah dikonversi ke urutan byte network dan sebaliknya.

Masih belum jelas? Bingung? Pegangan saja deh. Berita buruknya, konon saya bukan guru yang baik, jadi jika Anda belum jelas kemungkinannya dua, saya ndak bisa menjelaskan atau saya yang ndak ngerti. Ah entahlah. Sampai ketemu di bagian kedua. *Long Life and Prospher*! *

Catatan

* Salam bangsa Vulcan.