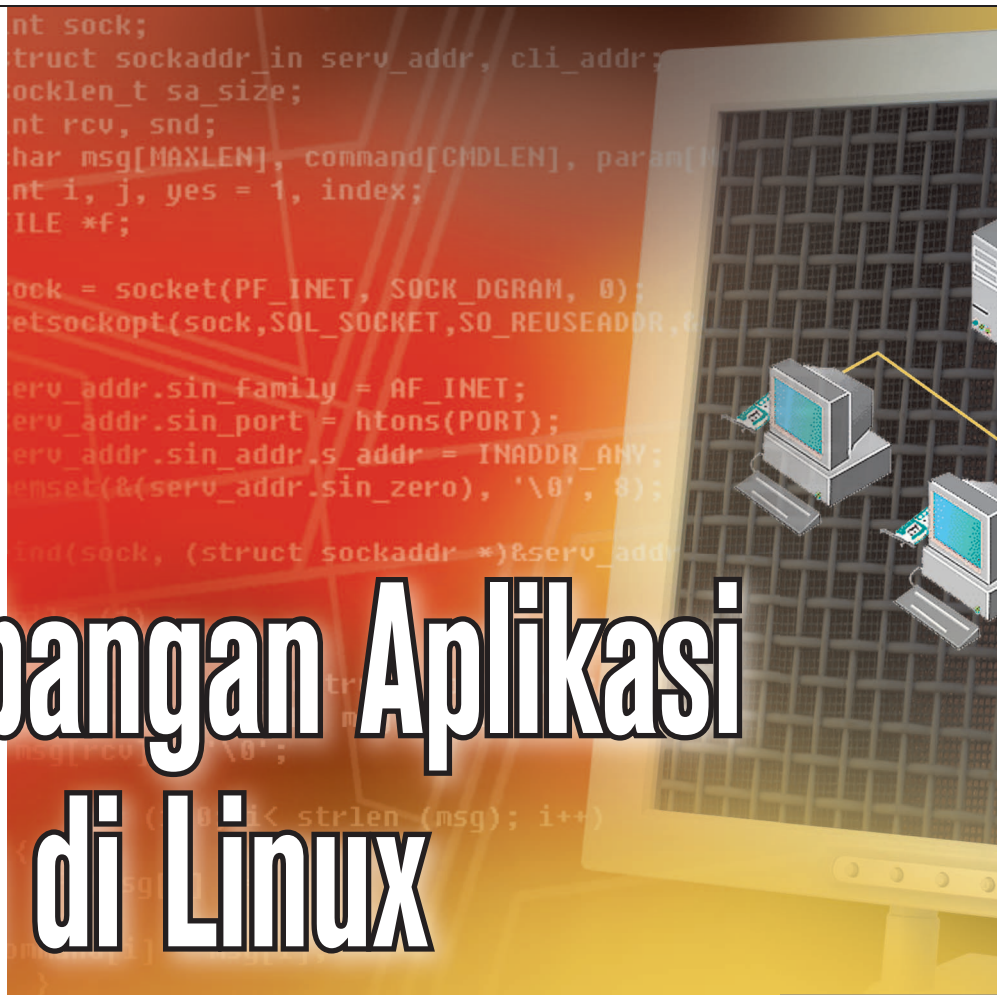
Noprianto

Di dunia *free software*, kita memiliki segala yang diperlukan untuk membangun aplikasi jaringan. Kita akan membahas beberapa di antaranya di dalam tulisan ini.



Pengembangan Aplikasi Jaringan di Linux

Jaringan adalah sesuatu yang hampir selalu kita gunakan dalam kehidupan sehari-hari menggunakan komputer, sadar ataupun tidak. Sebagai contoh adalah berselancar di Internet, berbagi file dengan rekan kerja melalui berbagai protokol *file sharing* yang tersedia, me-*remote* suatu *host*, bermain game multiplayer di jaringan lokal, dan lain sebagainya. Saat ini, cukup sulit untuk bekerja secara *offline*. Bayangkan jika Anda harus bertukar file dengan USB flash disk. Atau, ketika Anda harus membawa file ke suatu komputer secara manual agar dapat melakukan pencetakan.

Di dalam tulisan ini, kita akan membahas bagaimana kita membangun aplikasi jaringan di Linux. Aplikasi jaringan sederhana bisa berupa dua program yang saling berbicara satu sama lainnya dan kemudian mengambil tindakan sesuai pesan yang diterima dan dimengerti. Sebagai contoh, kita bisa membangun suatu server sederhana yang dapat mengartikan pesan yang dikirimkan kepadanya untuk melakukan tindakan tertentu, seperti melakukan perhitungan (protokol sederhana). Kita juga akan membangun aplikasi *client* yang bisa bicara sesuai bahasa yang dimengerti oleh server.

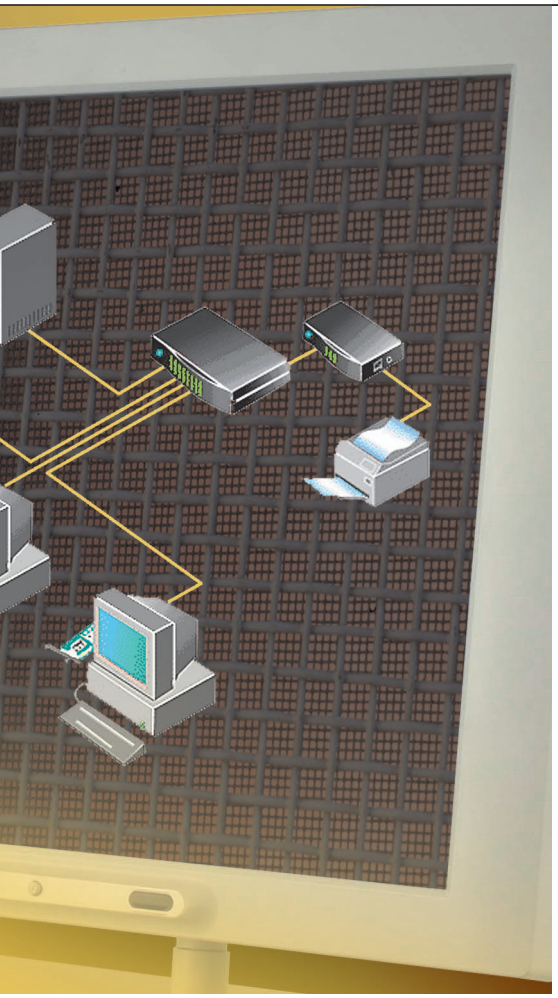
Selain protokol yang kita buat sendiri, di dunia ini terdapat sangat banyak protokol matang (kita batasi pada Application Layer), yang memanfaatkan jaringan untuk melakukan sesuatu. Sebagai contoh adalah HTTP (*HyperText Transfer Protocol*), FTP (*File Transfer Protocol*), SMB (*Server Message Block*), POP3 (*Post Office Protocol versi 3*), IRCP (*IRC Protocol*), DHCP (*Dynamic Host Configuration Protocol*), dan masih sangat banyak lagi. Pembahasan akan kita lakukan di antaranya dengan menghadirkan contoh client sederhana untuk salah satu protokol, yang dapat kita kembangkan sama-sama, baik di rubrik "Tutorial" ataupun di berbagai kesempatan lainnya.

Untuk membangun aplikasi jaringan, kita bisa mempergunakan berbagai bahasa pemrograman yang ada. Untuk bekerja dengan protokol-protokol populer, kita bisa mempergunakan berbagai pustaka yang telah tersedia, karena membangun client dari nol dengan memahami detail teknis protokol akan memakan waktu yang sangat lama.

Pembahasan akan kita mulai dengan contoh aplikasi client server sangat sederhana, dengan protokol kita sendiri. Kemudian, kita akan membahas secara sekilas berbagai pustaka yang tersedia untuk bekerja dengan berbagai protokol populer.

Setelah itu, kita akan membahas secara singkat protokol HTTP dan FTP. Sebuah contoh program HTTP client sederhana memanfaatkan pustaka yang ada juga akan kita bangun.

Terakhir, kita akan membahas berbagai aspek lain yang bisa diperhatikan ketika kita membangun aplikasi jaringan. Selamat membaca!



APLIKASI CLIENT SERVER SEDERHANA: REMOTE CALCULATOR

Di bagian ini, kita akan membahas bagaimana membangun aplikasi client server socket datagram berupa *remote calculator* sederhana, di mana terdapat sebuah server yang terus menerus bekerja dan mendengarkan pada port 27111. Di sisi lain, terdapat aplikasi client yang akan melakukan koneksi ke port tersebut dan berbicara di dalam bahasa yang dikenal oleh server.

Aplikasi akan dibangun dengan bahasa C menggunakan pustaka standar, baik untuk server ataupun client. Penekanan dalam pembahasan program ini adalah sepenuhnya kode-kode untuk bekerja dengan jaringan, di mana client dan server dapat saling berbicara. Untuk perhitungan matematika, kita akan menyerahkannya kepada python.

Beberapa catatan:

- Pemeriksaan kesalahan ketika pemanggilan berbagai system call dan pemeriksaan lain-lainnya tidak dilakukan. Dalam aplikasi di dunia nyata, ini merupakan hal yang mutlak. Pemeriksaan bisa dilakukan mulai dari pembuatan socket sampai penerimaan/pengiriman paket.
- Sebagai catatan, server akan melempar apa pun yang dikirimkan oleh client ke python. Artinya, bukan hanya ekspresi matematika saja. Perintah python pun akan dijalankan. Tentu saja, ini sangat berbahaya karena user mungkin mengirimkan kode-kode jahat. Di dunia nyata, Anda bisa mengimplementasikan pemeriksaan yang lebih teliti ataupun menggunakan cara lainnya untuk perhitungan.
- Secara teknis, eksekusi python kita lakukan dengan menggunakan fungsi popen().
- Python memiliki kemampuan perhitungan yang sangat baik. Di dalam aplikasi ini, kita membatasi panjang hasil perhitungan yang dikembalikan oleh program python.

Aplikasi server dan client akan bekerja pada modus teks. Untuk mempermudah, aplikasi client dan server akan dijalankan dalam satu komputer, di mana masing-masing client dan server akan dijalankan pada X terminal emulator yang berbeda.

Aplikasi akan kita berikan nama rem_calc.

Protokol

Protokol komunikasi di sini cukup sederhana.

- Setelah server dijalankan, maka client bisa dijalankan dan mengirimkan ekspresi matematika (diinput oleh user) diawali dengan tulisan CALC (contoh: CALC 1+2), dan server akan melakukan perhitungan dengan bantuan python, dan mengirimkan kembali kepada client dengan tulisan RESULT <hasil>.
- Perintah yang tidak dikenal akan diabaikan dan pesan kesalahan akan dikirimkan ke client. Saat ini, perintah yang dikenal hanyalah CALC. Antara perintah dan parameter, setidaknya dipisahkan oleh satu spasi.

Source code server

Berikut ini adalah source code server rem_calc, yang diberinama rem_calc_server.c:

```
/*
 * rem_calc_server
 * (c) nop, GPL.
 */

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

#define PORT 27111
#define MAXLEN 255
#define CMDLEN 20

int main(void)
{
    int sock;
    struct sockaddr_in serv_addr,
cli_addr;
    socklen_t sa_size;
    int rcv, snd;
    char msg[MAXLEN],
command[CMDLEN], param[MAXLEN],
result[MAXLEN];
    int i, j, yes = 1, index;
    FILE *f;

    sock = socket(PF_INET, SOCK_
DGRAM, 0);
    setsockopt(sock,SOL_SOCKET,SO_
REUSEADDR,&yes,sizeof(int));

    serv_addr.sin_family = AF_
INET;
    serv_addr.sin_port =
htons(PORT);
    serv_addr.sin_addr.s_addr =
INADDR_ANY;
    memset(&(serv_addr.sin_zero),
'\0', 8);

    bind(sock, (struct sockaddr
*)&serv_addr, sizeof(struct
sockaddr));

    while (1)
    {
      sa_size = sizeof(struct
sockaddr);
      rcv=recvfrom(sock, msg,
MAXLEN-1 , 0, (struct sockaddr
```

```
*)&cli_addr, &sa_size);
      msg[rcv] = '\0';

        for (i=0; i< strlen
(msg); i++)
      {
        if (msg[i] != ' ')
        {
       command[i] = msg[i];
        }
        else
        {
       index = i+1;
       break;

        }
      }
      command[i] = '\0';

      for (i = index, j=0; i<
strlen (msg)-1; i++, j++)
      {
     param[j] = msg[i];
      }
      param[j] = '\0';

      printf("[Received packet
from %s] %s",inet_ntoa(cli_addr.
sin_addr), msg);
      printf("\t Command is [%s]\
n", command);
      printf("\t Param is [%s]\n",
param);

      if (strcmp (command, "CALC")
== 0)
      {
     sprintf (msg, "python -c
'print %s'", param);

     f = popen (msg, "r");
     fgets (result, MAXLEN, f);
     pclose (f);

     result[strlen(result)-1] =
'\0';
     sprintf (msg, "RESULT %s",
result);
       printf("\t Response text
[%s]\n", msg);

     snd = sendto (sock, msg,
strlen (msg), 0, (struct sockaddr
*)&cli_addr, sizeof (struct
sockaddr));
      }
      else
      {
     strcpy (msg, "Unknown
command");
       printf("\t %s.\n", msg);
     snd = sendto (sock, msg,
strlen (msg), 0, (struct sockaddr
*)&cli_addr, sizeof (struct
sockaddr));
      }


    }

    close(sock);

    return 0;
}
```

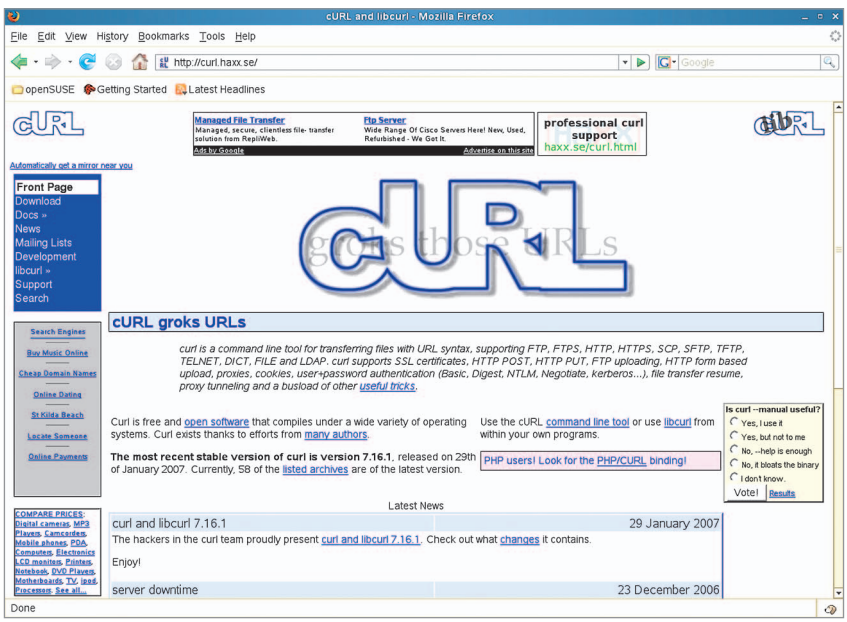
Untuk kompilasi, berikanlah perintah:

```
$ gcc -o rem_calc_server rem_calc_
server.c
```

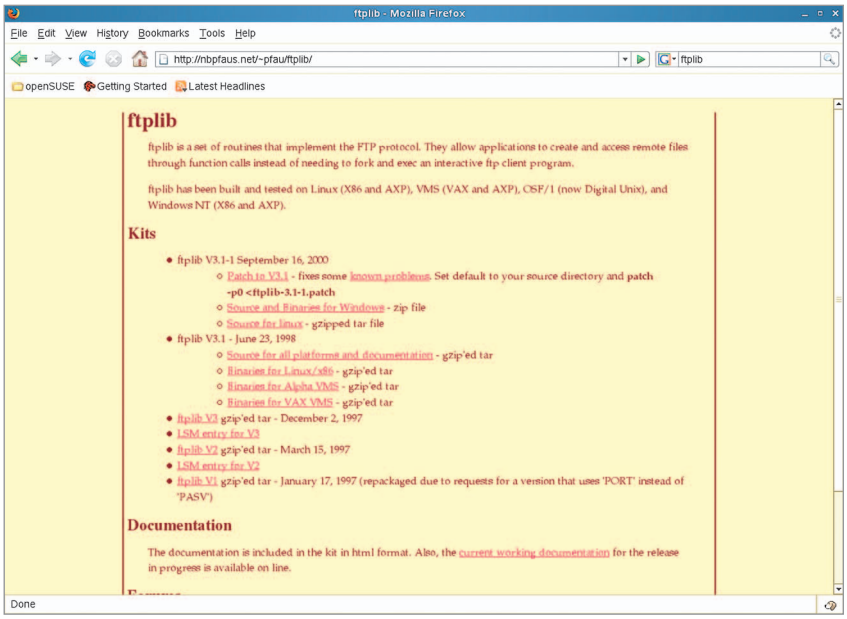
Source code client

Berikut ini adalah source code client rem_calc, yang diberinama rem_calc_client.c:

```
/*
 * rem_calc_client
 * (c) nop, GPL.
 */

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

#define PORT 27111
#define MAXLEN 255
#define SERV_ADDR "localhost"

int main()
{
    int sock;
    struct sockaddr_in serv_addr;
    struct hostent *h;
    int snd, rcv;
    char msg[MAXLEN];
    socklen_t sa_size;

    h = gethostbyname (SERV_ADDR);

    sock = socket(PF_INET, SOCK_
DGRAM, 0);

    serv_addr.sin_family = AF_
INET;
    serv_addr.sin_port =
htons(PORT);
    serv_addr.sin_addr = *((struct
in_addr *)h->h_addr);
    memset(&(serv_addr.sin_zero),
'\0', 8);
```



Situs web cURL.



Situs web ftplib.

```
   while ( 1 )
   {
   printf (“>> “);
   fgets (msg, MAXLEN-1, stdin);
   snd = sendto(sock, msg,
strlen(msg), 0, (struct sockaddr
*)&serv_addr, sizeof(struct
sockaddr));

   sa_size = sizeof (struct
sockaddr);
   rcv = recvfrom (sock, msg,
MAXLEN-1, 0, (struct sockaddr
*)&serv_addr, &sa_size);
   msg[rcv] = ‘\0’;
   printf (“%s\n”, msg);

   }

    close(sock);

    return 0;
}
```

Untuk kompilasi, berikanlah perintah:

```
$ gcc -o rem_calc_client rem_calc_
client.c
```

Menguji client dan server

Untuk menguji client dan server, lakukanlah langkah-langkah berikut:

- Bukalah satu x terminal dan jalankanlah aplikasi server:
  ```
  ./rem_calc_server
  ```
- Bukalah x terminal lain dan jalankanlah aplikasi client:
  ```
  ./rem_calc_client
  ```
- Di sisi client, sebuah prompt >> akan ditampilkan. Kita bisa mengirimkan ekspresi matematika dan server akan menampilkan hasil perhitungan input kita.

Contoh keluaran dari rem_calc_server:

```
$ ./rem_calc_server
[Received packet from 127.0.0.1]
CALC 4/2.2
        Command is [CALC]
        Param is [4/2.2]
        Response text [RESULT
1.81818181818]
[Received packet from 127.0.0.1]
CALC 12345 * 99999
        Command is [CALC]
        Param is [12345 * 99999]
        Response text [RESULT
1234487655]
[Received packet from 127.0.0.1]
CALC 123 ** 99
        Command is [CALC]
        Param is [123 ** 99]
        Response text [RESULT
795437446969572962215989474440348791896860112560759634932135019790296271625618826772704693920663805377147975587971037697856066197182782519397377584138716051409888241865404815003470901166310031944013830701187]
```

Contoh keluaran dari rem_calc_client:

```
$ ./rem_calc_client
>> CALC 4/2.2
RESULT 1.81818181818
>> CALC 12345 * 99999
RESULT 1234487655
>> CALC 123 ** 99
RESULT 795437446969572962215989474
4034879189686011256075963493213501
9790296271625618826772704693920663
3805377147975587971037697856066197
1827825193973775841387160514098882
4186540481500347090116631003194401
3830701187
>>
```

PUSTAKA YANG TERSEDIA (CLIENT)

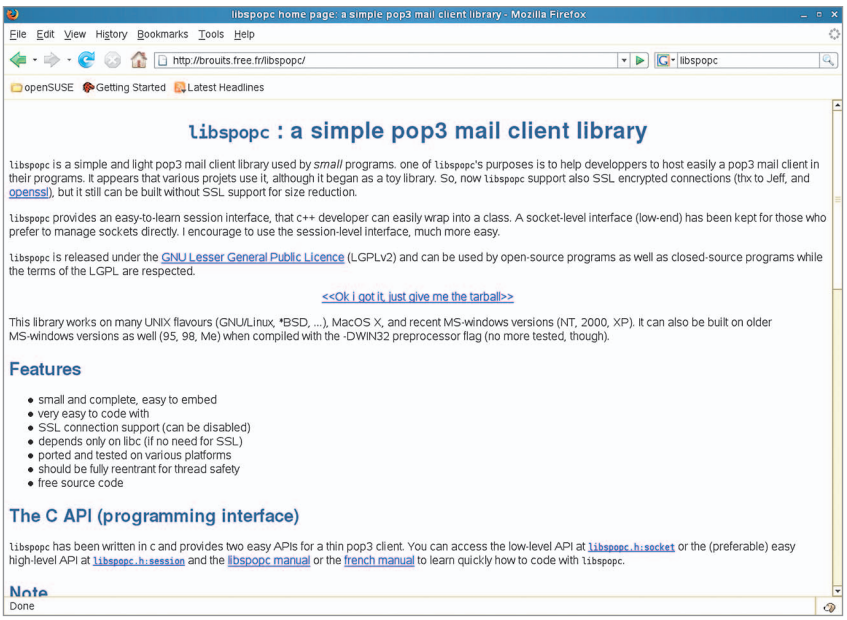
Dalam contoh sebelumnya, kita bisa melihat bahwa kita mencoba untuk membangun protokol sederhana, milik kita sendiri. Berbagai protokol yang ada jelas jauh lebih rumit daripada apa yang kita buat. Walau demikian, untuk berbagai protokol yang rumit sekalipun, selama spesifikasinya tersedia, kita bisa saja membaca dan menerapkan client sendiri. Namun, dengan spesifikasi puluhan halaman atau lebih, dengan segala kompleksitasnya, untuk berbicara dengan server saja, kita membutuhkan waktu yang relatif lama. Belum lagi masalah keamanan. Dan, ini tentu saja belum termasuk mengurusi fungsi utama aplikasi client.

Untungnya, di dunia ini, terdapat cukup banyak pustaka, yang memungkinkan Anda untuk berbicara dengan banyak protokol tanpa harus repot-repot memahami detail protokol tersebut. Kita akan membahas beberapa di antaranya.

Libcurl

Boleh dikatakan, ini adalah salah satu pustaka yang paling lengkap dan kaya fitur. Libcurl, yang bisa di-*download di http://curl.haxx.se/*, mendukung berbagai protokol berikut ini:

- FTP/FTPS (termasuk FTP upload).
- HTTP/HTTPS (POST, PUT, *form upload*, proxy, tunnel, dan lain-lain).
- SCP/SFTP.
- TFTP.
- TELNET.



Situs web libspopc.

- DICT.
- FILE.
- LDAP.

Sebagai suatu pustaka, libcurl juga bahkan mendukung banyak platform berikut ini:

- Solaris.
- NetBSD.
- FreeBSD.
- OpenBSD.
- Darwin.
- HP UX.
- IRIX.
- AIX.
- Tru64.
- Linux.
- UnixWare.
- HURD.
- Windows.
- Amiga.
- OS/2.
- BeOS.
- Mac OS X.
- Ultrix.
- QNX.
- openVMS.
- Novell Netware.
- DOS.
- Dan lain sebagainya.

Selain mendukung banyak protokol dan sangat portable, libcurl juga mendukung SSL certificate dan berbagai fitur lainnya, seperti:

- thread safe.
- Ipv6.
- Cepat.
- Didokumentasikan dengan baik.

Untuk menggunakan libcurl, kita bisa mempergunakan bahasa C ataupun puluhan bahasa pemrograman lainnya (*binding* tersedia untuk lebih dari 30 bahasa pemrograman).

Versi terbaru libcurl dan tool curl pada saat tulisan ini dibuat adalah 7.16.1 (29 Januari 2007).

ftplib

ftplib merupakan pustaka yang mengimplementasikan protokol FTP. Pustaka ini mengizinkan aplikasi untuk mengakses serta membuat file remote dengan mudah.

Ftplib mendukung platform-platform berikut:

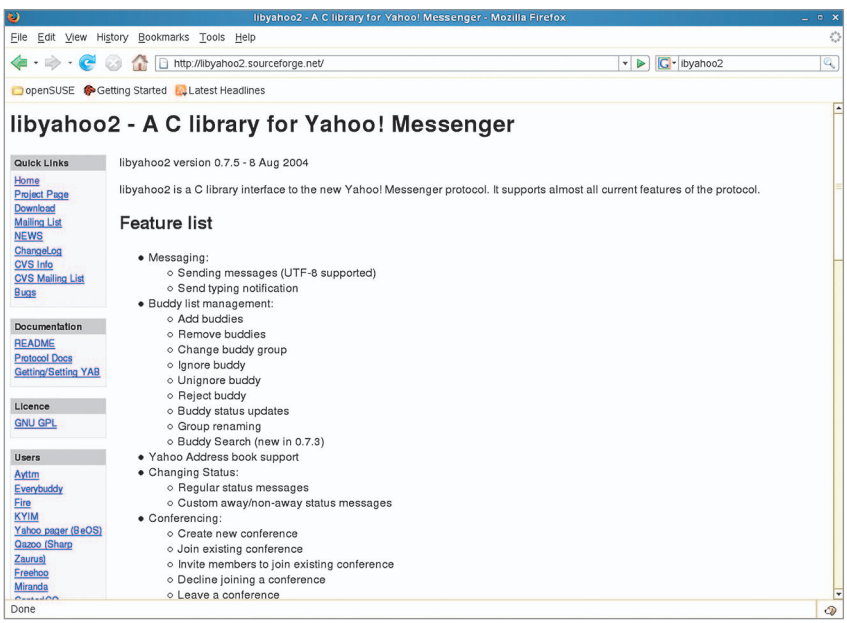
- Linux.
- VMS.
- DigitalUnix.
- Windows NT.

Pada saat tulisan ini dibuat, versi terbaru ftplib adalah 3.1-1, yang dirilis pada 16 September 2000.

Pustaka-pustaka lain

Berikut ini adalah contoh-contoh pustaka untuk berbagai protokol populer lainnya:

- libssh (untuk mengakses SSH service), *http://www.0xbadc0de.be/libssh:libssh*.
- libspopc (POP3 mail client), *http://brouits.free.fr/libspopc/*.
- libyahoo2 (Yahoo! Messenger protocol), *http://libyahoo2.sourceforge.net*.
- libirc (untuk mengakses IRC), *http://www.libirc.org*.
- UW IMAP (untuk bekerja dengan IMAP), *http://www.cac.washington.edu/imap/*.
- omniORB (CORBA), *http://omniorb.sourceforge.net*.
- Dan lain sebagainya.



Situs web libyahoo2.

PROTOKOL HTTP DAN FTP

Di bagian ini, kita akan membahas sekilas dua protokol yang cukup popular, yaitu HTTP dan FTP.

HTTP

HTTP adalah singkatan dari HyperText Transfer Protocol, yang merupakan metode yang digunakan dalam transfer informasi di WWW. HTTP merupakan protokol *request/response* antara client dan server. Di antara server dan client, terdapat kemungkinan penengah, seperti proxy, gateway dan tunnel.

Contoh HTTP server populer adalah Apache HTTP server. Contoh HTTP client populer adalah *web browser* seperti Mozilla Firefox. HTTP client sering pula disebut sebagai HTTP user agent.

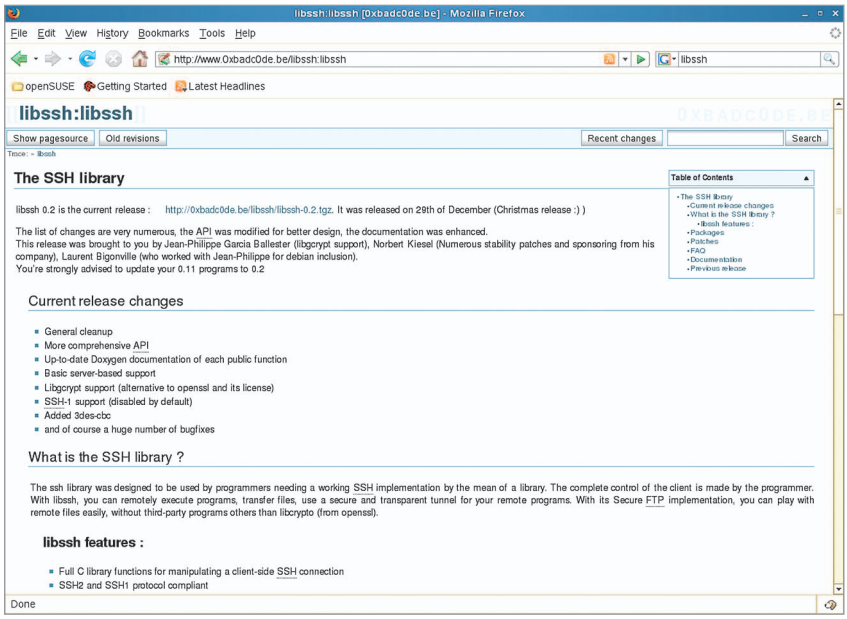
Versi protokol HTTP yang umum digunakan saat ini adalah HTTP/1.1 (Juni 1999).

Secara default, server HTTP akan membangun koneksi TCP dan membuka port 80, kemudian menunggu request dari client. Setelah menerima request, server akan mengirimkan status, pesan tambahan, isi *body* yang diminta, dan informasi lainnya.

Resource yang diakses oleh HTTP diidentifikasikan dengan URI http:.

Tanpa menggunakan user agent modern, kita pun dapat merequest informasi kepada HTTP server. Contoh caranya adalah dengan menggunakan program telnet untuk melakukan koneksi ke port 80 dan memberikan perintah seperti GET /index.html HTTP/1.0. Contoh:

```
$ telnet localhost 80
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
GET /index.html HTTP/1.0

HTTP/1.1 200 OK
Date: Tue, 20 Feb 2007 05:04:41
GMT
Server: Apache/2.2.3 (Unix)
mod_ssl/2.2.3 OpenSSL/0.9.8d DAV/2
PHP/5.2.0
X-Powered-By: PHP/5.2.0
Content-Length: 44
Connection: close
Content-Type: text/html

<html><body><h1>It works!</h1></
body></html>Connection closed by
foreign host.
```



Situs web libssh.

FTP

FTP adalah singkatan dari File Transfer Protocol, yang digunakan untuk menghubungkan dua komputer sehingga pengguna di komputer satu dapat mentransfer file ke/dari komputer lainnya, termasuk memberikan perintah operasi file di komputer lainnya tersebut.

Komputer yang menyediakan layanan disebut sebagai FTP server (membutuhkan program ftp server). Untuk melakukan koneksi ke FTP server, dibutuhkan *software* ftp client.

Secara default, FTP server akan membangun koneksi TCP dan membuka port 21 untuk *incoming connection* dari client (membentuk control stream). Untuk transfer file aktual, koneksi yang berbeda diperlukan (data stream). Pengaturan data *stream* tergantung transfer mode:

- Pada mode aktif, ftp client akan membuka port acak (di atas 1023), mengirimkan informasi port tersebut kepada ftp server, dan menunggu koneksi dari server. Pada saat FTP server menginisiasi data connection, bind ke port 20 server (source port) akan dilakukan.
- Pada model pasif, ftp server akan membuka port acak (di atas 1023), mengirimkan informasi port tersebut kepada ftp client dan menunggu koneksi dari client. Dalam hal ini, bind source port (oleh client) akan dilakukan untuk port acak di atas 1023.

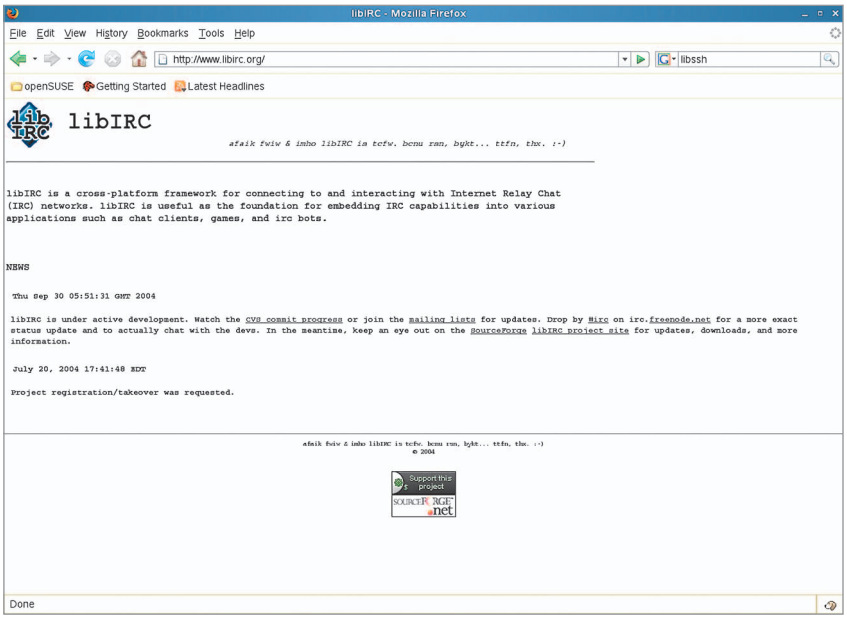
Saat ini, terdapat sangat banyak ftp client (contoh: gFTP). Berbagai web browser populer, selain berfungsi sebagai HTTP client, juga berfungsi sebagai FTP client, walaupun mungkin tidak datang dengan fitur lengkap untuk FTP.

CLIENT HTTP DENGAN LIBCURL

Berikut ini adalah contoh HTTP client sederhana, memanfaatkan pustaka libcurl. Fungsi utama program ini adalah mendownload URL tertentu. Hanya men-download saja, tanpa fitur tambahan (seperti *resume*, *multithreading*, dan lain sebagainya).

Source code program http_client_simple.c:

```
/*
 * simple_http_client
 * (c) Nop, GPL.
 */

#include <stdio.h>
#include <curl/curl.h>

int main (int argc, char * argv[])
{
        CURL * curl;
        CURLcode res;
        FILE * f;
```



Situs web libirc.

```
        if (argc != 3)
        {
                fprintf (stderr,
"usage: simple_http_client <URL>
<output_file>\n");
                exit (1);
        }

        f = fopen (argv[2], "w");

        curl = curl_easy_init();
        if (curl)
        {
                curl_easy_setopt
(curl, CURLOPT_URL, argv[1] );
                curl_easy_setopt
(curl, CURLOPT_WRITEDATA, f);
                res = curl_easy_
perform (curl);

                fclose (f);
                curl_easy_cleanup
(curl);
        }

        return 0;
}
```

Cara kompilasi:

```
$ gcc -o http_client_simple http_
client_simple.c -lcurl
```

Cara menjalankan:

```
$ ./http_client_simple
usage: simple_http_client <URL>
<output_file>
```

Contoh:

```
$ ./http_client_simple http://
localhost/index.html out.html
$ cat out.html
<html><body><h1>It works!</h1></
body></html>
```

```
$ ./http_client_simple http://
localhost/apache_pb.gif out.gif
$ file out.gif
out.gif: GIF image data, version
89a, 259 x 32
```

ASPEK LAIN YANG PERLU DIPERHATIKAN

Berikut ini adalah beberapa hal yang barangkali bisa menjadi pertimbangan, dalam konteks pengembangan aplikasi jaringan.

Platform

Untuk protokol yang standar dan matang, baik client ataupun server umumnya memiliki implementasi yang berjalan pada berbagai platform, mulai dari platform komputer besar sampai desktop. Kadang-kadang, untuk aplikasi client, malah terdapat implementasi yang berjalan pada perangkat *mobile*.

Ketika mengimplementasikan suatu protokol, ada baiknya kalau kita mampu membangun client ataupun server yang berjalan pada berbagai platform (multiplatform). Untuk contoh kasus server, lihatlah pada Apache HTTP server, web server paling banyak digunakan di dunia ini, yang mampu berjalan pada berbagai platform. Untuk contoh kasus client, lihatlah pada mozilla suite yang juga mampu berjalan pada sangat banyak platform.

Ketika membangun suatu protokol, dimana merupakan hal yang sangat rumit, ada baiknya kalau kita mampu membangun protokol yang cukup netral dan tidak memihak platform tertentu, apabila memungkinkan.

Bagaimanapun juga, ketika kita bekerja dengan jaringan, terdapat kemungkinan banyak platform saling bekerja sama. Ini sangatlah alami, seperti halnya kehidupan.

Kebutuhan pustaka

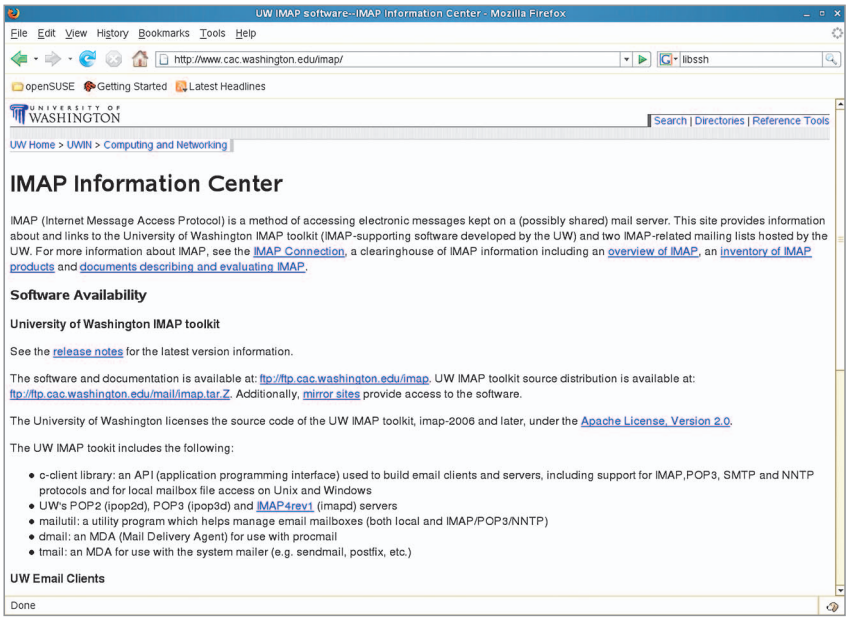
Idealnya, kita mempergunakan berbagai pustaka yang telah ada di dunia free software ini. Dengan demikian, kita tidak menulis ulang fungsionalitas yang telah ada. Tenaga yang ada bisa disumbangkan untuk penyempurnaan pustaka tersebut, ataupun membangun yang belum ada. Kita pun bisa lebih fokus pada fungsi utama program. Namun, harus diperhatikan juga kondisi dimana kita hanya membutuhkan sebagian kecil dari suatu pustaka berukuran besar (yang tidak terlalu modular).

Dan, tentunya, dalam memilih pustaka, tentukan juga kriteria yang sesuai dengan kebutuhan. Untuk satu fungsionalitas, terkadang, terdapat beberapa pustaka tersedia. Pustaka-pustaka yang ada tersebut memiliki nilai plus minus masing-masing. Salah memilih pustaka bisa berdampak cukup besar pada program akhir.

Bahasa pemrograman

Bahasa pemrograman apa yang harus dipergunakan? Ini tentunya sangat bergantung kepada Anda sendiri, selaku sang programmer. Hanya, pertimbangkanlah berbagai hal berikut:

- Kecepatan. Ketika Anda membutuhkan kecepatan tinggi, menggunakan bahasa C (compiled) barangkali lebih cocok dibandingkan dengan menggunakan bahasa Python (*interpreted*).



Situs web UW IMAP.

- Ruang kosong. Ketika Anda membangun untuk sistem *embedded* atau sistem kecil, maka ruang kosong menjadi sangat penting. Kalau Anda membangun menggunakan Perl atau Python, Anda membutuhkan puluhan mega byte harddisk. Ini belum ditambah pustaka eksternal. Kalau Anda menggunakan C, Anda mungkin hanya memerlukan ratusan KB (kompilasi statis) atau beberapa MB (kompilasi dinamis, telah memperhitungkan C library dan pustaka pendukung).

Keamanan

Dalam aplikasi jaringan, keamanan merupakan hal yang sangat penting. Dengan membuka satu port, kita telah membuka pintu ke komputer kita. Kita tidak ingin membangun aplikasi server yang mudah jebol hanya karena tindakan DoS sederhana. Begitupun dengan masalah alokasi memory. Atau aspek-aspek lainnya.

Salah satu indahnya dunia free software adalah saling bantu antar-developer di seluruh dunia. Ketika kita merilis server kita sebagai free software, semakin banyak pihak yang bisa mengaudit kualitas program kita. Dan, tentunya, juga cukup banyak yang bisa membantu memperbaiki kelemahan yang ada.

Multithreading dan anak proses

Ketika server Anda menjadi cukup serius dan digunakan oleh banyak client, maka barangkali perlu dipikirkan juga tentang threading ataupun anak proses. Tergantung pada teknik yang lebih Anda sukai.

Dalam membangun aplikasi client, sebagai contoh HTTP downloader, tak jarang teknik memanfaatkan anak proses ataupun memanfaatkan multithreading juga digunakan.

Hanya, untuk isu seperti ini, developer diharapkan untuk sangat peduli terhadap isu perbedaan platform. Kode-kode untuk bekerja dengan anak proses atau multithreading bisa berbeda antara satu dengan lainnya. Bahkan, tidak semua fitur tersedia pada setiap platform.

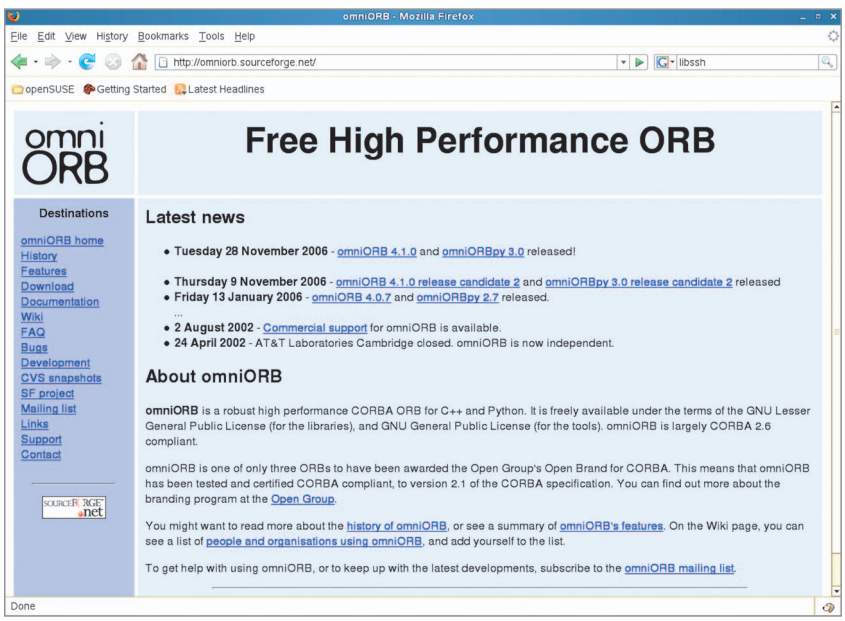
Front end

Apabila memungkinkan, pisahkan antara user interface grafikal/text-based dengan fungsi utama program. Fungsi utama program, bisa berbentuk program biasa ataupun pustaka, hendaknya dikembangkan dengan sangat matang dan teliti. Selanjutnya, user interface bisa dikembangkan oleh siapa saja.

Dengan memisahkan antara core program dengan *front end*, lebih mudah bagi developer lainnya untuk membangun user interface yang disukai. Bisa berbasis grafikal atau text based atau command line, tergantung pada developer dan target pengguna.

Selain itu, dengan memisahkan core dan front end, kita juga bisa berkonsentrasi pada masing-masing hal tersebut dengan lebih baik lagi.

Tulisan ini hanya membahas sangat sedikit dari dunia pemrograman jaringan di Linux. Banyak sekali hal yang bisa di-*explore* di dunia free/open source software. Ratusan pustaka, ratusan milis, ribuan developer dan masih banyak lagi siap membantu kita membangun program yang kita inginkan. Setelah itu, kita pun bisa membantu yang lain. Demikian seterusnya. *Happy hacking*!



Situs web omniORB.