PENGGUNAAN PROXY SEBAGAI ACCOUNT MANAGER DALAM SUATU ISP

Nama : Tri Widodo S.B
NIM : L2F300568
Jurusan Teknik Elektro Fakultas Teknik
Universitas Dionegoro Semarang

ABSTRAK

Selama beberapa tahun ini perkembangan sistem komunikasi berkembang sangat pesat. Pada jaringannya juga harus memiliki sistem yang bagus, untuk melakukan suatu komunikasi dimana keamanan data yang dimiliki terjamin. Penggunaan lalulintas data dalam internet dewasa ini terasa semakin banyak diperlukan. Penggunaan jalur komunikasi ini juga harus diautr, karena kebebasan berkoneksi inilah yang membuat kepadatan jalur lalulintas Internet. Dengan melakukan suatu pengaturan dalam berkomunikasi akan lebih meningkatkan kualitas penggunaan koneksi Internet yang semakin bijaksana, karena client hanya akan berhubungan dengan server yang benar-benar dibutuhkan. Yang menjadi permasalahan utama adalah bentuk dari pengaturan dalam berkoneksi dan pengaturan jalur lalulintas Internet.

Dalam Tugas Akhir ini, dibuat jenis pengaturan yang berupa filter alamat client dan filter data hasil koneksi dari server.

I. PENDAHULUAN

Latar Belakang

Perkembangan dunia komputer semakin pesat. Apalagi sekarang telah didukung dengan fasilitas Internet yang canggih. Dengan adanya Internet maka, akan sangat membantu untuk menambah informasi dan berkomunikasi. Perkembangan teknologi yang pesat sudah mewarnai kehidupan manusia sekarang. Komputer yang digunakan sekarang saja tidak mungkin digunakan sebagai komputer pribadi. Banyak orang membutuhkan banyak akan informasi dan berkomunikasi maka, perlu adanya integrasi komputer sehingga membentuk suatu jaringan.

Tujuan

Tujuan yang akan dicapai dengan dibuatnya tugas akhir ini adalah membuat perangkat lunak *web proxy* menggunakan pemrograman *socket* dengan memakai bahasa pemrograman C untuk mengatur lalulintas data pada koneksi Internet.

Pembatasan Masalah

Melihat dari luas dan kompleksnya permasalahan yang ada pada komununikasi *client-server* Internet ini, maka diperlukan batasan-batasan untuk menyederhanakan permasalahan yaitu:

1. Perangkat Lunak dibatasi hanya untuk menangani jenis komunikasi *client/ server* yang ada pada *protocol HTTP* Internet.
2. Dalam hal ini pembahasan masalah difokuskan pada *proxy* generik yang digunakan sebagai *account manageryang mengatur client* yang berkoneksi, jumlah byte yanag dimiliki *client* untuk berkoneksi, dan jumlah byte data maksimal setelah terjadi koneksi ke *server.*
3. Pembahasan Tugas Akhir ini dibatasi masalahnya yaitu mengenai koneksi *client server* menggunakan identitas pengguna atau alamat asal *client.*

II. Landasan Teori

2.1 Protokol HTTP

Protokol HTTP *(Hyper Text Transfer Protocol)* adalah suatu protokol yang digunakan untuk pengiriman data di *World Wide Web (WWW)*. Protokol ini mengirimkan data-data yang berbentuk teks, *hypertext*, audio, video, dan lainnya. Untuk format pengiriman pesan yaitu dari *client* ke *server* dan dari *server* ke *client*. Format pesan HTTP tidak dapat dibaca oleh pengguna, karena dikirim dan diperintah oleh HTTP *server* dan *client* HTTP (*browser*). Konsep dari HTTP adalah suatu *client* mengirim permintaan ke *server*. *Server* akan menerima, dan kemudian akan mengirim kembali data yang diminta oleh *client* [4].

2.1.1 Transaksi HTTP

Client adalah suatu program yang mengirimkan suatu permintaan hubungan dengan sebuah *server*, yang kemudian akan mendapatkan suatu tanggapan dari *server* yang dihubungi. Sebuah *client* dapat berkomunikasi dengan *server* baik secara langsung maupun tidak langsung. Mereka dapat berkomunikasi sewaktu-waktu, dengan menggunakan media tranmisi yang telah ada. Satu *server* bisa berhubungan lebih dari satu *client*. Tugas *server* menyediakan pelayanan untuk para

client, dan juga menghubungkan *client* dengan para pengguna jasa komunikasi yang lain. Tetapi dalam praktiknya, sebuah *server* sulit untuk berkomunikasi dengan *client* yang cukup banyak.

Meskipun HTTP menggunakan pelayanan protokol TCP, hubungan yang terjadi berbeda. *Client* mengirim permintaan, *server* mengirim kembali sebagai tanda hubungan.

2.1.2 Pengiriman Pesan

Suatu baris permintaan terbagi atas tipe permintaan, URL, dan versi HTTP. Penulisan baris permintaan terdiri atas tipe permintaan, spasi, URL, spasi dan versi HTTP. Tipe Permintaan Menjelaskan macam tipe dari bentuk permintaan – permintaan dari *client*. *Client* yang akan mengakses sebuah alamat *Web* membutuhkan sebuah alamat. Fasilita yang dibutuhkan untuk mengakses dokume seluruh dunia, HTTP menggunakan konsep pengalokasian. URL merupakan suatu standar spesifikasi untuk suatu informasi pada suatu Internet. Versi HTTP merupakan versi yang digunakan. Dalam suatu URL terdapat beberapa bagian yaitu : metode, *host*, komputer, port dan *path*.

a. Metode
   Metode adalah suatu protokol yang digunakan untuk menempatkan suatu dokumen. Beberapa protokol lain yang digunakan untuk penempatan suatu dokumen adalah diantaranya : Gopher, FTP, HTTP, News, dan TELNET.
b. Host
   Host adalah komputer dimana suatu informasi itu berada, meskipun nama dari komputer itu sendiri dimisalkan. Halaman *Web* biasanya berada di komputer, dan komputer memberikan nama pengganti yang biasanya diawali dengan karakter "WWW". URL dapat diberisi nomor *port* sebuah *server*.
c. Port
   Penulisan nomor port oleh *client* digunakan untuk penunjukkan suatu alamat yang akan diakses. Penggunaan nomor port ini akan membantu *client* untuk koneksi ke *server*. Jika port dimasukkan, port berada diantara host dan *path* yang dipisahkan oleh sebuah tanda titik dua.
d. Path
   Path adalah bagian dari nama sebuah arsip informasi berada. Sebagai catatan dalam Penulisan *path* terdapat garis miring dengan sendirinya, dalam sistem operasi UNIX untuk memisahkan direktori dan subdirektori.

2.1.3 Respon Pesan

Dalam respon pesan terdapat *status line*, *header* dan *body*.

a. Versi HTTP
   Bentuk HTTP sama dengan bentuk di baris permintaan.
b. Kode Status
   Bentuk dari kode status lebih mendekati bentuk protokol FTP dan SMTP, yang terdiri dari tiga digit. Kodenya terbagi atas jarak 100 yang bersifat informasi, jarak 200 kode sebagai indikasi permintaan *client* telah berhasil. Jarak 300 kode menghubungkan langsung *client* dengan URL lain, jarak 400 kode mengindikasikan proses gagal pada sisi *client*, dan jarak 500 kode sebagai indikasi proses gagal pada sisi *server*.

4

c. Frase Status
   Pada bentuk ini menjelaskan kode status dalam bentuk teks, karena kode status yang sebelumnya menuliskan tentang tiga digit angka yang menjelaskan kode status dari suatu proses.

2.1.4 Header

Header melakukan penukaran informasi antara *client* dan *server*. Sebagai contoh *client* dapat meminta pengiriman dokumen dalam format khusus atau *server* mengirimkan informasi sekilas mengenai dokumen. *Header* dapat menjadi satu atau lebih *header line*. Baris *header* memiliki empat katagori : *header* umum, *header* permintaan, *header* tanggapan, dan *header* satuan.

a. Header Umum
   Header yang memberikan informasi akurat mengenai pesan dan dapat dilakukan bersamaan antara permintaan dan tanggapan.
b. Header Permintaan
   Header permintaan hanya dapat dijalankan pada permintaan pesan. Didalamnya terdapat penjelasan mengenai konfigurasi *client* dan bentuk format dokumen dari *client*. Tabel 2.3 menunjukkan bentuk dari *request header*.
c. Header Tanggapan
   Header tanggapan hanya dapat ditunjukkan pada tanggapan pesan. Menunjukkan konfigurasi *server* dan informasi khusus mengenai permintaan *client*
d. Header Satuan
   Header satuan memberikan informasi mengenai *body* dari dokumen. Meskipun lebih banyak digunakan di tanggapan pesan, beberapa permintaan pesan, seperti metode POST atau PUT, tetapi juga

menggunakan *header* tipe ini. Tabel 2.4 menunjukkan bentuk dari *header* satuan.

2.2 Protokol Jaringan

Client yang membutuhkan informasi dari sebuah *server* haruslah melalui sebuah jaringan komputer. Dalam proses penyampaian informasi ini mengalami suatu proses. Informasi yang dibutuhkan *client* diubah menjadi data–data yang diolah menjadi segmen-segmen. Segmen-segmen inipun akan diolah lagi menjadi sebuah paket, yang diolah lagi menjadi sebuah *frame*. Kemudian proses terakhir *frame* menjadi sebuah bit, yang dapat dikirim melalui kabel dalam suatu jaringan ke komputer lain untuk diproses balik seperti informasi semula [4,2].

2.2.1 Referensi Model OSI

Untuk mempermudah proses pengiriman data ini dibutuhkan suatu peralatan jaringan. Pengelolaan jaringan ini membutuhkan suatu lapisan jaringan. *ISO (International Standard Organization)* mengeluarkan suatu model lapisan jaringan yang disebut *OSI (Open System Interconnection)*. Dalam model *OSI* tidak membahas secara detail cara kerja dari setiap lapisannya, melainkan memberikan suatu konsep bagaimana proses terjadi, protokol yang dipakai disuatu lapisan tertentu. Tabel 2.5 menunjukkan lapisan model *OSI* beserta fungsi dan protokol yang melayani.

2.2.2 Model TCP/IP

Selain model *OSI*, terdapat juga model yang lain yaitu model TCP/IP. Model ini dikeluarkan oleh *DOD (Departement Of Defense Amerika)*, yang berperan penting dalam pembuatan dasar-dasar hubungan Internet.

TCI/IP adalah jenis protokol yang pertama yang digunakan dalam hubungan Internet. Tetapi kebanyakan jenis protokol inilah yang sering dipakai oleh orang kebanyakan. Hal ini yang menyebabkan Protokol jenis TCP/IP digunakan sebagai standar *de facto*, yaitu standar yang diterima karena pemakaian yang meluas dengan sendirinya.

Lapisan Proses merupakan gabungan dari lapisan aplikasi, presentasi dan sesi. Protokol-protokol yang berfungsi di lapisan ini adalah :

a. Telnet
   Telekomunication Network berguna untuk seorang pemakai komputer untuk mengakses komputer lain dari jauh. Telnet menggunakan port 23 untuk berhubungan dengan lapisan transport.
b. File Transfer Protocol (FTP)
   Protokol ini berfungsi untuk memindahkan file dari satu komputer ke komputer yang lain lewat jaringan. FTP disini menggunakan port 21 dan bekerja dengan protokol TCP yang menggunakan hubungan *connection oriented* . Umumnya hubungan dengan FTP berupa relasi antara *client server*. FTP *server* harus dibangun dahulu pada suatu jaringan, agar komputer *client* dapat berhubungan.
c. Simple Mail Transfer Protocol (SMTP)
   Protokol ini berfungsi untuk mengatur pengiriman electronic mail. SMTP menggunakan port 25, bekerjanya mirip dengan FTP menggunakan protokol TCP.

Lapisan Host To Host merupakan sama dengan lapisan transport yang memungkinkan data dikirim tanpa kesalahan. Lapisan ini terdiri atas :

a. Transmission Control Protocol (TCP)
   Protokol ini berfungsi untuk mengubah blok data yang besar menjadi suatu segmen-segmen yang dinomori dan disusun secara berurutan.
b. User Datagram Protokol
   Protokol ini termasuk jenis *connection oriented*, yang bergantung pada lapisan diatasnya untuk mengontrol keutuhan data. Karena penggunaan bandwidth yang efektif, UDP digunakan untuk aplikasi yang tidak peka terhadap gangguan jaringan. TCP dan UDP menggunakan port 1023 atau lebih besar untuk berhubungan dengan host dari lapisan atas.

Lapisan Internet merupakan lapisan yang menentukan jalur pengiriman dan meneruskan paket ke alamat peralatan lain yang berjauhan

a. Internet Protokol (IP)
   Internet Protokol adalah protokol yang memberikan alamat atau identitas logika untuk peralatan di jaringan. IP address disebut alamat logika karena dibuat oleh perangkat lunak, dimana alamat tersebut secara dinamis dapat berubah jika ditempatkan di jaringan yang lain.
b. Address Resolution Protocol (ARP)
   ARP adalah protokol yang mengadakan translasi dari alamat IP yang diketahui menjadi *hardware address.*
c. Dynamic Host Configuration Protocol (DHCP)
   Protokol ini dapat memberikan alamat IP secara otomatis ke suatu *workstation* yang menggunakan protokol TCP/IP. DHCP ini bekerja pada suatu relasi *client server*, dimana *server* dapat

menyediakan suatu alamat IP kepada suatu *client.*

Lapisan Network Access merupakan gabungan lapisan data link dan lapisan fisik model OSI disebut lapisan bawah yang oleh model DOD disebut lapisan *network access*.

2.3 Socket

Protokol yang banyak digunakan secara umum adalah TCP/IP. Orang banyak menggunakan protokol ini karena mempunyai sifat *open networking*, artinya penjelasan secara teknis dibahas dengan jelas. Sedangkan *interface* yang digunakan oleh LINUX adalah *Socket* BSD. *Socket* yang dimaksud adalah bukan sebuah perangkat keras, melainkan perangkat lunak yang digunakan pada saat komunikasi LINUX. *Socket* INET yaitu *socket* yang berfungsi pada saat pengalamatan berlangsung. *Socket* ini juga mengatur komunikasi protokol TCP/IP [3,10].

2.4 Konfigurasi Client Server

Client adalah suatu program yang mengirimkan suatu permintaan hubungan dengan sebuah *server*, yang kemudian akan mendapatkan suatu tanggapan dari *server* yang dihubungi [4]. Sebuah client dapat berkomunikasi dengan *server* baik secara langsung maupun tidak langsung. Mereka dapat berkomunikasi sewaktu-waktu, dengan menggunakan media transmisi yang telah ada. Satu *server* bisa berhubungan lebih dari satu *client*. Tugas *server* disini menyediakan pelayanan untuk para *client*, dan juga menghubungkan *client* dengan para pengguna jasa komunikasi yang lain.

2.5 Proxy

Proxy adalah suatu sistem pengaturan dan pembatasan lalu lintas Internet berdasarkan suatu aturan tertentu dari Internet (*public network*) ke *private network*, demikian pula sebaliknya dari *private netwrok* ke Internet. Biasanya komputer yang berperan sebagai *proxy* terletak di antara *private network* dengan *public network* (Internet). Semua lalu lintas paket data dilewatkan melalui komputer ini sehingga komputer ini dapat menerapkan peraturan yang dikonfigurasikan kepadanya. *Proxy* dapat bersifat terbuka jika dikonfigurasikan untuk mengizinkan hampir semua servis Internet bebas keluar masuk dari jaringan. Tetapi juga bisa bersifat tertutup jika dikonfugurasikan dengan peraturan yang ketat dan akses yang sangat terbatas [5].

III. Perancangan Sistem

Perancangan perangkat lunak yang dilakukan adalah pengaturan koneksi *client server* yang lebih teratur. Dalam kondisi koneksi umum *client* biasa koneksi ke *server* dengan alamat tujuan yang bebas dengan waktu yang tak terbatas dan pengambilan data sesuai dengan keinginan *client*. Tetapi dalam hal ini terdapat suatu *proxy* yang membatasinya. Dalam Tugas Akhir ini pembahasan masalah hanya mencangkup koneksi *client* ke *server* menggunakan IP *client* pengguna yang koneksi ke *server* dan jumlah byte data yang diambil oleh *client.*

3.1 Penggunaan Fungsi Forking pada Program

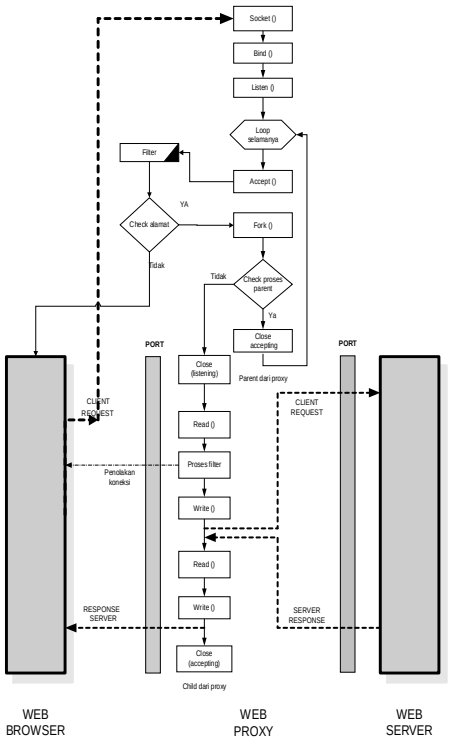
Fungsi *fork()* pada program *proxy* didesain untuk mampu menangani beberapa proses dari program. Fungsi *fork()* ini jika berhasil akan menciptakan satu proses anak (*child process*) yang identik dengan proses *parent* sebelumnya (artinya mempunyai semua sifat dari proses *parent*). Setelah *fork()* berhasil, yaitu dengan mengecek nilai balik tidak sama dengan nol (fork() != 0) maka proses *parent* akan menutup *socket accept*, sehingga dapat segera mendengarkan kembali pada koneksi dari *web browser (listening)* [10].

Proses *child* akan mengambil alih koneksi dari *web browser*. Pada tahap ini proses *child* akan menutup *socket listening* yang tidak dibutuhkan lagi karena kegiatan ini adalah kegiatan yang dilakukan oleh proses *parent*. Proses *child* yang akan menangani proses pertukaran data dari *web browser* dan menuju ke *web server*.

3.2 Proses Awal Koneksi

Pada awal koneksi *client* melakukan permintaan ke sebuah *server*, permintaan ini merupakan permintaan koneksi HTTP, yang berisi alamat yang diinginkan oleh *client.* Kemudian *server* ini akan melakukan pembacaan terhadap koneksi tersebut, pembacaan ini diberikan pada bagian filter dari *proxy* untuk identifikasi terhadap alamat yang dikehendaki oleh *web browser*.

Gambar 3.1 menunjukkan diagram alir *web proxy*, terlihat mulai dari proses *client request*, diterima oleh *proxy* untuk filter alamat kemudian sampai *respond* dari *server* yang diterima kembali oleh *client*.



Gambar 3.1 Diagram Alir Web Proxy

Proses koneksi awal adalah sebagai berikut:

1. *Web server* yang dipakai adalah A*pache web server* komputer. *Client* dapat melakukan koneksi ke *web browser* dengan IP *address* atau alamat tujuan dengan bebas misalnya 127.xxx.xxx.xxx. Sedangkan untuk alamat localhost sendiri yaitu "127.0.0.1" (alamat IP untuk *localhost* pada system operasi Linux) digunakan sebagai alamat dari *client*.
2. *Socket*, Perintah *socket* ini digunakan untuk membuat *socket* sesuai dengan permintaan. jika terjadi kegagalan dalam pemanggilan sehingga pesan *error* akan ditampilkan di layar atau *console*, dan jika pemanggilan berhasil, maka nilai baliknya akan bernilai positif.

   Socket ini mempunyai tiga argumen dan kembaliannya berupa interger :

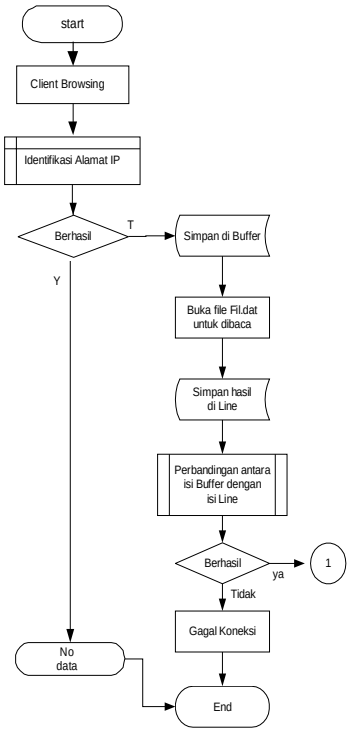
   Int *socket* (af, type, protocol)

   Argumen af yang menunjukkan sistem pengalamatan. Argumen type ini menentukan komunikasi yang dipakai, misalnya jenis TCP atau UDP. Argumen protokol digunakan untuk mendukung *socket* dalam suatu pengalamatan.
3. *Bind*, setelah *socket ()* berhasil diciptakan, dan alamat dan *port* nya telah diisi, kemudian *proxy* memanggil fungsi *bind*() untuk melakukan penambatan *socket* tersebut. Jika *return value* bernilai positif, maka pemanggilan *bind* berhasil, namun jika nilai balik lebih kecil dari 0 (<0), maka terjadi *error* sehingga pesan *error* akan ditampilkan pada layar.
4. *Listen()*, proses selanjutnya adalah memanggil fungsi *listen()* sehingga *socket* segera menunggu koneksi dari *web proxy*. *Socket* dibatasi dengan jumlah maksimal koneksi yang dapat ditangani sekaligus. Jika jumlah koneksi dari *web browser* telah mencapai jumlah maksimum (= BACKLOG), maka koneksi yang datang setelahnya akan ditolak. Fungsi ini akan mengembalikan nilai negatif (<0), jika terjadi error dalam proses listen.
5. *Accept()*, fungsi ini segera dipanggil ketika masuk koneksi dari *web browser*. Jika proses *accept* ini berhasil, suatu *socket* baru akan tercipta, sehingga pada saat ini sisi *server* telah memiliki dua buah *socket*. Fungsi ini juga akan mengembalikan nilai negatif (<0) jika terjadi kegagalan.

3.3 Proses Filter IP Client/ Alamat Asal

Pada bagian ini, semua koneksi *client* yang masuk akan diperiksa dan kemudian akan disaring. Semua alamat asal *client* akan dibandingkan dengan alamat yang ada pada *proxy*.

1. Akses yang masuk diterima dengan menggunakan perintah *accept*, maka *client* dapat koneksi. Pada *proxy* akan tertulis alamat yang dituju oleh *client* dan alamat cilent itu sendiri.
2. Dengan menggunakan alamat asal atau user ID ini maka *client* dapat dikontrol dalam koneksi alamat tujuan dan pengambilan data. Jika alamat *client* diblokir atau tidak berhasil maka koneksi berikutnya akan gagal karena alamat client akan dibandingkan dengan alamat yang ada pada data *proxy*.
3. Pengiriman data yang dilakukan, akan tertulis dan tersimpan dalam buffer. Dalam penyimpanan buffer akan menjelaskan isi dari hasil koneksi yang terjadi diantaranya berapa jumlah byte dan format file apa yang digunakan.
4. Langkah berikutnya adalah membuka file alm.dat dengan fungsi *fopen()* yang berisi daftar user ID dan jumlah byte client.



Gambar 3.2 Proses filter alamat client

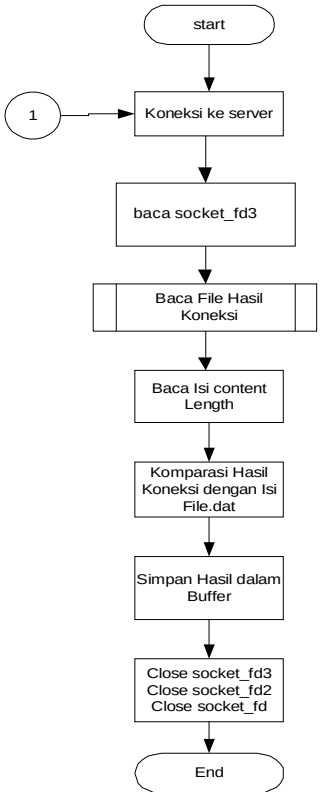
5. Segera setelah dibuka, kemudian isi file dibaca dengan fungsi *fgets()* dan hasil pembacaan disimpan di tempat penyimpanan sementara.
6. Hasil pembacaan isi dari kedua tempat penyimpanan inilah yang akan dibandingkan dengan memakai fungsi *strstr()*. Fungsi ini akan mengembalikan isi argumen kedua (pembanding) atau NULL jika tidak ada yang sama.
7. Apabila filtering untuk alamat yang dituju dapat lolos, maka koneksi akan berlanjut lagi yang artinya proses filtering alamat telah dilakukan.
8. *Proxy* kemudian segera melakukan pembacaan terhadap *socket* hasil *accept* dengan menggunakan fungsi *read()*. Hasil pembacaan ini akan disimpan pada *buffer* dengan ukuran *buffer* sebesar MAXBUF. Fungsi ini juga akan mengembalikan nilai negatif (-1), jika tejadi kegagalan dalam pembacaan.
9. Pada file alm.dat telah tertulis alamat *client*, sehingga nantinya tidak semua *client* dapat *request* /mengakses data tanpa melalui proses filtering. Dari sini juga dilihat *client* yang *request* memiliki jumlah byte dapat digunakan untuk melakukan koneksi ke *server*.
10. *Proxy* memanggil fungsi *socket* lagi untuk membuat *socket* baru yang akan dipakai untuk melakukan koneksi ke *web server*. Jika pemanggilan ini berhasil, selanjutnya akan dilakukan pengisian alamat tujuan beserta *port* tujuannya.
11. *Proxy* kemudian akan memanggil fungsi *connect()* untuk melakukan koneksi kepada *web server*. Fungsi ini akan mengembalikan nilai –1 jika terjadi kegagalan dan mengembalikan nilai 0 jika pemanggilan berhasil, sehingga komunikasi dengan *web server* dapat segera dilakukan.
12. *Proxy* akan segera memanggil fungsi *write*() untuk melakukan penulisan permintaan dari *web browser*, selanjutnya sisi *client* ini akan menunggu respon dari *web server*, fungsi *write* akan mengembalikan nilai negatif jika terjadi kegagalan baca, dan akan mengembalikan jumlah data (dalam byte) yang ditulis (dikirim) jika berhasil.
13. Jika respon dari *web browser* tiba, maka *proxy* ini akan memanggil fungsi *read*() untuk membaca respon tersebut dan tanpa perlu melakukan apapun terhadap hasil pembacaannya, maka hasil pembacaan ini akan segera diteruskan kepada *web browser* dengan memanggil fungi *write*(). Fungsi *read()* ini juga akan mengembalikan nilai negatif (-1) jika terjadi kegagalan baca, dan jika berhasil maka fungsi ini akan mengembalikan jumlah data (dalam byte) yang berhasil dibaca.

3.4 Proses Filter Data

Gambar 3.3 menunjukkan proses filter data dimana pengaturan jumlah byte yang diambil oleh *client* dari hasil download yang telah dibatasi jumlah maksimalnya oleh *proxy*. Ketika jumlah byte maksimal terpenuhi maka secara otomatis koneksi akan berhenti.

1. Kemudian setelah respon dari *web browser* yang berupa data yang merupakan permintaan dari *client* terkirim, langkah berikutnya adalah membuka file kembali alm.dat dengan fungsi fopen() yang berisi jumlah byte maksimal yang dapat diakses oleh browser. Dalam pembukaan file alm.dat yang kedua adalah untuk memeriksa jumlah byte maksimal yang boleh diambil dari suatu *server* oleh koneksi client.
2. Segera setelah dibuka, kemudian isi file dibaca dengan fungsi fgets() dan hasil pembacaan disimpan di tempat penyimpanan sementara.
3. Hasil pembacaan isi dari kedua tempat penyimpanan inilah yang akan dibandingkan dengan memakai fungsi strstr(). Fungsi ini akan mengembalikan isi

argumen kedua (pembanding) atau NULL jika tidak ada yang sama.



Gambar 3.3 Proses Filter Data

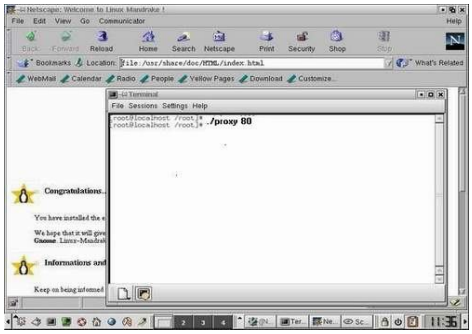
4. Karena di dalam file alm.dat telah terdapat batasan byte maksimal yang boleh diambil, maka setiap pengambilan data secara otomatis jumlah yang kita tentukan akan dikurangi oleh hasil koneksi client dengan kata lain hasil browsing yang berlangsung.
5. Selama koneksi berlangsung jumlah byte yang telah ditentukan akan terus berkurang sampai mencapai batasannya, maka koneksi yang sedang berlangsung akan berhenti. Pada saat jumlah byte yang ada telah mencapai batas minimal maka akan tertulis "ERROR CONNECTION" pada web server. Apabila ingin mengambil data lagi, maka client diharuskan melakukan koneksi baru lagi.
6. Jika sudah tidak terjadi koneksi lagi, maka tugas bagian filter selesai, dan selanjutnya sisi client dari proxy yang akan melakukan tugas terakhir dari proxy.
7. Setelah komunikasi ini selesai dilakukan, maka proses akan kembali berulang ke awal proses lagi, dan seterusnya, hingga tidak ada lagi permintaan dari *web browser* (dengan diterimanya tanda pemutusan koneksi dari *web browser*).

Dengan demikian semua kegiatan socket akan berhenti karena tidak ada lagi koneksi berikutnya. Penjelasan tersebut mengakhiri semua kegiatan yang ada pada proses koneksi dari *client* ke *server* dengan melewati proxy untuk proses filtering.

IV. Analisis dan Pengujian

4.1 Pengaktifan Program

Dalam percobaan ini port *server* dalam hal ini apache web *server* menggunakan port yang telah ada dalam hal ini port 80 atau port default. Sedangkan pada koneksi yang dilakukan oleh *client* port yang digunakan tidak ditentukan secara langsung. Untuk menjalankan program *proxy* yang telah dibuat dengan menuliskan : #./*proxy* 80, maka secara langsung koneksi yang dilakukan dengan port tersebut. Untuk melihat program telah berjalan menggunakan perintah *#netstat –a –t*, sehingga dapat dihasilkan suatu daftar *socket* TCP yang sedang mendengarkan koneksi beserta nomor port.



Gambar 4.1 Pengaktifan Program Proxy

Maka dengan sendirinya program tersebut akan berjalan pada port 80, seperti yang terlihat pada Gambar 4.1. Jika *client* browsing ke suatu server, *client* menuliskan ke alamat mana yang akan dituju beserta nomor port. Secara ototmatis, koneksi *client server* berjalan sesuai dengan yang diharapkan.

4.1.1 Penggunaan IP Address

Dalam melakukan suatu koneksi antar komputer pada TCP, dibutuhkan 2 macam identifikasi, yaitu alamat IP (IP address) dan nomor *port* pada keduanya seperti yang terlihat pada Gambar 4.2. Kombinasi dari kedua identifikasi tersebut disebut sebagai alamat *socket*. Untuk menggunakan pelayanan TCP membutuhkan sepasang alamat *socket*, yaitu alamat *socket client* dan alamat *socket server*. Alamat *socket client* akan menunjukkan

program aplikasi pada *client* secara unik, juga demikian halnya dengan alamat *socket server*, akan menunjukkan program aplikasi pada *server* secara unik. Keempat komponen ini merupakan bagian dari *header* IP dan *header* TCP. *Header* IP memuat alamat IP sedangkan *header* TCP memuat nomor *port*.

Gambar 4.2 Tampilan Koneksi *Cient Server* Berjalan

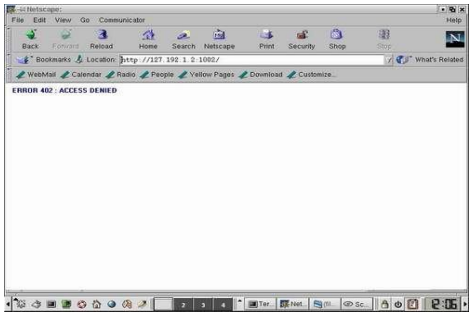
4.1.2 Penulisan Nomor Port

Penulisan nomor port dalam Tugas Akhir oleh *client* digunakan untuk penunjukkan suatu alamat yang akan diakses. Penggunaan nomor port ini akan membantu *client* untuk koneksi, sedangkan untuk *server* sendiri membantu dalam filtering alamat. Untuk penentuan alamat port yang digunakan dalam Tugas Akhir ini adalah *port default* yaitu 80. Port ini biasa digunakan untuk komunikasi protokol HTTP. Karena sistem operasi yang digunakan adalah Linux, untuk itu *web browser* yang digunakan salah satunya adalah Netscape.

4.2 Pengujian Filter IP Client/ Alamat Asal

Pada pengujian program ini, sebelum *client* mengambil data pada browser harus melakukan penulisan alamat tujuan terlebih dulu. Pertama kali akan muncul identifikasi diri pada *proxy* yang berupa alamat asal atau IP *client*. Pada saat akses terjadi maka proses filter secara otomatis berjalan. Apabila alamat yang *client* berada dalam daftar tidak boleh diakses, maka permintaan gagal dan *client* tidak dapat melanjutkan koneksi berikutnya ke sebuah *server* atau *request* data selanjutnya. Kegagalan koneksi akan terus berlanjut apabila alamat IP *client/*asal terdapat dalam daftar yang tidak boleh diakses. Apabila terjadi kesamaan pada waktu perbandingan maka, koneksi akan gagal dan akan ditolak oleh program dengan menuliskan pesan: "ERROR 402: ACCESS DENIED!" pada *web browser*. Jika alamat tujuan lolos proses filter, koneksi akan berlanjut sesuai yang diinginkan oleh *client*.

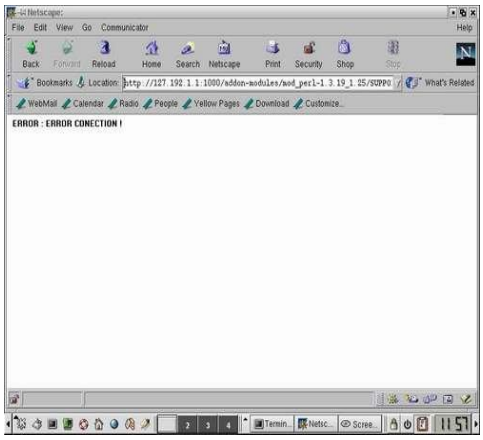
Gambar 4.3 menunjukkan tampilan pada layar *web browser* saat terjadi tolak koneksi dari *client request* ke *server* tidak lolos proses filter alamat.



Gambar 4.3 Tampilan Tolak Koneksi oleh Proxy

4.3 Pengujian Filter Data oleh Proxy

Proses filter berikutnya adalah filter IP *client* yang melakukan browsing memiliki jumlah byte untuk pengambilan data ke *server*. Program diakses menggunakan *web proxy* yang membandingkan jumlah byte yang terdapat pada *file* alm.dat dengan jumlah byte data yang diambil oleh *client*. *Client* tidak bisa melebihi jumlah byte yang telah diberikan, karena dengan inilah *client* dapat dibatasi koneksi ke sebuah *server*. Proses pengurangan jumlah byte atau manajemen byte berlaku sampai jumlah byte yang telah disediakan untuk akses data telah habis atau kurang. Apabila jumlah byte yang dimiliki telah mencapai batas minimum maka pada *web browser* akan bertuliskan "ERROR CONNECTION". Dengan demikian koneksi yang berlangsung akan berhenti dan *client* tidak bisa melanjutkan akses data selanjutnya. Dalam program sudah terdapat program penghitung byte, sehingga setiap byte data yang diakses akan langsung mengurangi jumlah byte yang telah disediakan dalam file. Hal ini akan terus menerus berlangsung sehingga jumlah byte yang disediakan habis. Untuk dapat mengakses lagi *client* harus melakukan login ulang sehingga nantinya akan disediakan lagi jumlah byte untuk akses ke *server*. Gambar 4.4 menunjukkan koneksi yang berakhir karena jumlah byte yang dimiliki oleh *client* untuk *request* telah habis.



Gambar 4.4 Tampilan Akhir Koneksi dari Filter Byte Client

Dalam pengujian ini, ternyata dapat dibuktikan bahwa penggunaan *proxy* sebagai *account manager* berhasil. Hal ini setelah melakukan perbandingan antara jumlah byte dalam file check.dat dengan jumlah byte hasil akses, alamat yang diterima untuk melakukan akses melakukan koneksi. Pada saat koneksi dilakukan maka *proxy* akan mencatat jumlah data yang diambil, format data, tanggal pengambilan data, dan waktu pengambilan data. Pada file ini tidak tampilkan keseluruhan karena pada program ini hanya dititik beratkan pada proses pembatasan akses *client* ke *server*. Diharapkan pada tahap berikutnya dapat dikembangkan lagi, sehingga nantinya akan lebih sempurna program account manager ini.

V. PENUTUP

5.1 Kesimpulan

Dari hasil perancangan dan implementasi serta analisa, maka dapat diambil kesimpulan sebagai berikut:

1. Proses filtering untuk IP *client* berhasil karena IP *client* yang terdapat daftar tidak boleh melakukan koneksi akan ditolak koneksinya.
2. Program *web proxy* dapat melakukan proses filtering terhadap pengambilan data dari *web server* jika melebihi jumlah maksimal maka koneksi berakhir.

5.2 Saran

Berdasarkan perancangan, hasil pengujian dan analisa maka dapat disarankan hal-hal berikut untuk peningkatan/ perbaikan, antara lain:

1. Untuk penolakan akses secara langsung oleh *web proxy* dapat dilakukan dengan penambahan password dan penulisan identitas. Password pertama kali akan keluar pada saat *client* membuka jendela *web browser*.
2. Jika proses penyimpanan data setelah terjadi koneksi yaitu proses account itu kendalanya dapat terpecahkan.

DAFTAR PUSTAKA

1. Comer, Douglas, *Internetworking With TCP/IP: Principles, Protocols, and Architectures, Volume I,* Prentice-Hall Englewood Cliff-New Jersey, 1988.
2. Donahoo, Michael J. & Calvert, Kenneth L., *The Pocket Guide to TCP/IP Sockets, C Version*, Morgan Kaufmann Publisher, 2001.
3. Forouzan, Behrouz A., *TCP/IP Protocol Suite,* McGraw-Hill, Inc., 2000.
4. Hartono, Jogiyanto, *Konsep Dasar Pemrograman Bahasa C, Edisi Kedua,* ANDI Yogyakarta, 2000.
5. Jhony H. Sembiring, *Jaringan Komputer Berbasis LINUX*, Jakarta, 2002.
6. Kernighan, Brian W. & Ritchie, Dennis M., *The C Programming Language, 2nd Edition,* Prentice-Hall Inc., 1988.
7. Petersen, Richard, *Red Hat Linux: The Complete Reference*, Osborne-McGraw-Hill, 2000.
8. W. Richard Stevens, *UNIX Network Programing, Volume I, Second Edition,* Prentice-Hall PTR, 1997.

Mengetahui/ menyetujui :

Pembimbing I

Wahyudi, ST, MT
NIP. 132 086 662

Pembimbing II

Agung BP, ST, MIT
NIP. 132 137 932