ネットワークアーキテクチャ

レポート

2020531902 鈴鹿真世

[b0531902@eng.kitakyu-u.ac.jp](mailto:b0531902@eng.kitakyu-u.ac.jp)

1. **HTTPに関する説明**

HTTPとはWebクライアントとWebサーバーの間でハイパーテキストを転送するための通信プロトコルのことである．HTTPの仕組みとして，WebクライアントがWebサーバーにHTMLファイルを要求し，Webサーバーが要求されたHTMLファイルをWebクライアントに返すことでブラウザにWebページが表示される． この時にWebクライアントがWebサーバーにHTMLファイルといった要求をリクエスト，WebサーバーがWebクライアントに応答することをレスポンスと呼ぶ．要求されるデータはHTMLファイルだけでなく，音声，動画といったファイルもメタ情報を含めてやり取りすることが可能だ．HTTPリクエストおよびレスポンスは要求や返答の内容，資源の種類や形式などの情報，および関連する情報を記述した ヘッダ部と，送受信する資源などの本体であるボディ部で構成される．ボディ部は基本的にはレスポンスに存在するが，クライアント側からデータを送信する際にはリクエストにも付加される．

HTTPメソッドとは，サーバ にどのような動作を要求するかを短い符号で表したもので，指定した資源の送信を求める「GETメソッド」，ヘッダのみを送信するように求める「HEADメソッド」，クライアントからのデータの送信を求める「POSTメソッド」，HTTPから送信したデータを指定した資源に保存するよう求める「PUTメソッド」，資源削除を求める「DELETE」などが定義されている．

HTTP応答とは，要求を受けたWebサーバがWebクライアントへ返す応答メッセージのことを言う．ヘッダにはWebクライアントからの要求に対する応答(ステータスコード)と，要求された資源についての属性情報(メタ情報)などが記述される．送信するファイルなどの種類やデータ形式，データ長など資源に関する情報や，Webサーバの名称やバージョンなどの情報が含まれる．

1. **プログラムに関する説明**

実際に私が書いたソースコードはレポートの末尾に付録する．Webサーバ側のプログラムでは，始めにsocket()関数でソケットを生成する．次にbind()関数でIPアドレスとポート番号を設定して，listen()関数でクライアント側とコネクションが出来るまで待つ．それから，クライアント側とコネクションが確立した時にHTTPの処理を行っている関数を呼び，後にHTTPレスポンスを行う関数も呼ぶ．

一方，Webクライアント側のプログラムは，コマンドラインからホスト名を取得し，URIの処理を行うことでホストの情報を取得する．そして，サーバー側とコネクションを確立してHTTPプロトコルを作り，サーバに取得したホスト情報を送信してサーバー側とデータの送受信を行っている．データの送受信をサーバー側と行うことでコマンドラインから指定したURIにブラウザからアクセスが出来た．どのような手順を踏んでブラウザにアクセスが出来たのか過程の説明をこれから行う．

　始めにクライアントとサーバーとの接続後はクライアントからのリクエストメッセージをサーバー側が受信する．続いてクライアントから受信したリクエストメッセージからどのようなリクエストであるかを解析する．そして，解析した結果に応じてサーバー側が処理を行う．なお，解析した結果とはメソッドやリクエストターゲットなどを指す．この時に，リクエストのメソッドがGETであれば，リクエストに応じるためにリクエストターゲットに対応するデータ(index.html)を読み込む．次に，クライアントにレスポンスするためにサーバー側がレスポンスメッセージを作成してsend()関数でクライアントに送信する流れになっている．これで１つのリクエストに対する処理の一連の流れが実行される．

したがって，クライアント側のプログラムを実行する時に./web\_clie <http://localhost:50000/index.html>というコマンドライン引数を指定する．それから，リクエスト側のプログラム上で<http://localhost:50000/index.html>のホスト名やポート番号などの処理を行ってリクエストメッセージを作成し，サーバー側に作成したリクエストメッセージを送信する．そして，クライアント側から送信されたリクエストメッセージに応じてサーバー側が処理を行い，クライアント側にレスポンスメッセージを送ることでindex.htmlのデータが読み込めてブラウザからアクセスが出来るようになったと考える．

1. **実行結果，考察**

図１にサーバー側の実行結果を示し，図2にはクライアント側の実行結果を示す．そして図３にはブラウザでアクセスした結果を示す．図１の実行結果よりサーバー側の方では既にクライアントとコネクションを確立する準備が出来ていることが分かる.

一方，図２の実行結果よりクライアント側ではサーバー側からのレスポンスが表示されている．HTTP/1.1より1コネクションで，複数回の要求，応答がやりとりされている．また，index.htmlのデータも表示されている．そのため，サーバー側でindex.htmlのデータを読み込めたことも分かった．



図１　サーバー側の実行結果

**![テキスト, 手紙

自動的に生成された説明]()**

図2　クライアント側の実行結果

**グラフィカル ユーザー インターフェイス, テキスト, アプリケーション

自動的に生成された説明**

図３　ブラウザでアクセスした結果

1. **感想**

今回，C言語によるソケットプログラミングの演習を行った．クライアント側とサーバー側のそれぞれの処理の流れを理解することはできた．また，授業中に配布されたレジュメだけでなく，インターネット上に上がっているサンプルプログラムも動かし，ソケット通信の基礎的な部分を学ぶことが出来た．

しかし，資源削除を求める「DELETE」といった他のメソッドやヘッダ情報などに対応することは出来なかった．そのため，今回の演習を踏まえてヘッダ情報にも対応出来るようなプログラミングを書いてブラッシュアップを行いたい．

1. **参考文献**

[1] HTTPサーバプログラムの作成 (<http://research.nii.ac.jp/~ichiro/syspro98/wwwserver.html>)

[2] C言語でHTTPクライアントを作ってみよう(1) (<http://x68000.q-e-d.net/~68user/net/c-http-1.html>)

[3] [C言語] 簡単な「HTTPサーバーを作る」(<https://daeudaeu.com/c-http-server/#i-10>)

1. **付録**

web\_serv.c (サーバー側のプログラム)

#include<netinet/in.h>

#include<arpa/inet.h>

#include<strings.h>

#include<sys/fcntl.h>

#include<sys/socket.h>

#include<sys/types.h>

#include<unistd.h>

#include<stdlib.h>

#include<stdio.h>

#include<netdb.h>

#define PORT 50000

//HTTPの処理を行っている関数

void http(int sockfd){

int len;

int read\_fd;

char buf[1024];

char meth\_name[16];

char uri\_addr[256];

char http\_ver[64];

char \*uri\_file;

//sockfdはwebクライアントとの間の仮想通信路のソケット番号

if (read(sockfd, buf, 1024) <= 0 ) {

printf("error: reading a request.\n");

}

else {

sscanf(buf, "%s %s %s", meth\_name, uri\_addr, http\_ver);

if (strcmp(meth\_name, "GET") != 0) {

send\_msg(sockfd, "501 Not Implemented");

}

else{

uri\_file = uri\_addr+1;

//指定ファイルがあったら開く

if ((read\_fd = open(uri\_file, O\_RDONLY, 0666)) == -1) {

send\_msg(sockfd, "404 Not Found");

}

else {

send\_msg(sockfd, "HTTP/1.1 200 OK\r\n");

send\_msg(sockfd, "text/html\r\n");

send\_msg(sockfd, "\r\n");

while((len = read(read\_fd, buf, 1024)) > 0) {

if (write(sockfd, buf, len) != len) {

printf("error: writing a response.\n");

break;

}

}

close(read\_fd);

}

}

}

}

//HTTPレスポンスを行う関数

int send\_msg(int fd, char \*msg) {

int len;

len = strlen(msg);

if ( write(fd, msg, len) != len ){

printf("error: writing.");

}

return len;

}

int main(){

int sockfd;

int connectfd;

struct sockaddr\_in addr;

socklen\_t addrlen;

struct sockaddr\_in serv;

struct hostent \*host;

char buff[65000];

//ソケットを生成

sockfd = socket(PF\_INET,SOCK\_STREAM,0);

//IPアドレス　ポート番号の設定

bzero((void\*)&serv,sizeof(serv));

serv.sin\_family=AF\_INET;

serv.sin\_addr.s\_addr=htonl(INADDR\_ANY);

serv.sin\_port=htons(PORT);

if(bind(sockfd,(struct sockaddr \*)&serv,sizeof(serv))<0){

fprintf(stderr,"bind error\n");

exit(1);

}

//コネクション待受

listen(sockfd,10);

printf("connecting...\n");

//コネクション確立

addrlen = sizeof(addr);

while(1){

if((connectfd=accept(sockfd,(struct sockaddr \*)&addr,&addrlen))<0){

fprintf(stderr,"accept error/n");

break;

}

else{

http(connectfd);

close(connectfd);

}

}

close(sockfd);

}

web\_clie.c　(クライアント側のプログラム)

#include<netinet/in.h>

#include<arpa/inet.h>

#include<strings.h>

#include<sys/types.h>

#include<sys/socket.h>

#include<unistd.h>

#include<stdlib.h>

#include<stdio.h>

#include<netdb.h>

char buff[65000];

#define PORT 50000

int main(int argc,char \*argv[]){

int sockfd;

int port;

struct sockaddr\_in serv;

struct hostent \*host;

char hostname[65536];

char path[65536];

char \*p;

//コマンドラインからホスト名を取得

//URLの処理例より

if(sscanf(argv[1],"http://%s",hostname)<=0){

printf("Address error\n");

};

//ホスト名から"/"までを抜き出す

p=strchr(hostname,'/');

//パスを取得

strcpy(path,p);

\*p='\0';

p=strchr(hostname,':');

port = atoi(p+1);

if ( port <= 0 ){

port = 80; //ポート番号を80にする

}

\*p='\0';

printf("host:%s\npass:%s\nport num:%d\n",hostname,path,port);

/\*ホストの情報を取得\*/

struct hostent \*servhost;

servhost=gethostbyname(hostname);

if(servhost==NULL){

printf("IP addless error :%s",hostname);

return 0;

}

/\* 構造体をゼロクリア \*/

bzero(&serv, sizeof(serv));

serv.sin\_family = AF\_INET;

/\* IPアドレスを示す構造体をコピー \*/

bcopy(servhost->h\_addr, &serv.sin\_addr, servhost->h\_length);

serv.sin\_port = htons(port);

/\*ソケットの生成 socket()\*/

sockfd = socket(AF\_INET,SOCK\_STREAM,0);

/\*コネクション確立 connect()\*/

if(connect(sockfd,(struct sockaddr \*)&serv,sizeof(serv))<0){

fprintf(stderr,"connect error\n");

exit(1);

}

/\* HTTP プロトコル生成 & サーバに送信 \*/

sprintf(buff, "GET %s HTTP/1.1\r\nHost: %s\r\n\r\n", path, hostname);

send(sockfd, buff, strlen(buff),0);

/\*データの送受信\*/

while(1){

int size = recv(sockfd, buff, sizeof(buff) - 1, 0); // 受信

if(size > 0){

buff[size] = '\0';

printf("%s", buff); // 表示

}else{

break;

}

}

/\*ソケットの開放\*/

close(sockfd);

}