ネットワークアーキテクチャ
レポート

2020MCA011　後藤 悠志
b0mca011@eng.kitakyu-u.ac.jp

HTTP に関する説明

HTTP は，Web サーバと Web クライアント間において Web コンテンツを送受信するために用いられるプロトコル（通信規約）である．基本的にはテキストメッセージの交換を行い，Web ページを構成する HTML ファイルや，ページに関連づけられたスタイルシート，スクリプト，画像，音声，動画といったファイルを，メタ情報を含めてやり取りすることができる．プル型通信を基本とし，クライアントから要求を送り，サーバが応答を返す．クライアントからの要求およびサーバの応答は，要求や返答の内容，資源の種類や形式といった情報などを記述したヘッダ部と，送受信するファイルといった資源の本体であるボディ部で構成される．

HTTP メソッドは，サーバにどのような動作を要求するかを表すものであり，指定した資源の送信を求める「GET メソッド」，ヘッダのみ送信するよう求める「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」などがある．

HTTP 応答は，要求を受けたサーバがクライアントへ返す応答メッセージである．ヘッダにはクライアントからの要求に対する応答と，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，データ長といった資源に関する情報や，サーバの名称やバージョンといった情報が含まれる．

プログラムに関する説明

レポート末尾に Web サーバおよび Web クライアントのソースコードを付録する．

Web クライアントについて，はじめに *socket*() 関数によりソケットの生成を行う．続いて，URI のホスト名，ポート番号，パスに分ける処理を行ってから，*connect*() 関数によりコネクションの確率を行う．コネクション確率後，*send*() 関数により URI のそれぞれをサーバ側に送信し，サーバからの応答の受信体制に入る．

Web サーバについて，クライアントと同様にはじめに *socket*() 関数によりソケットの生成を行う．続いて，*bind*() 関数，*listen*() 関数により，IP アドレス・ポート番号の設定およびコネクション待ち受けを行い，*accept*() 関数によりコネクションを確立する．クライアント側から URI 情報を *recv*() 関数により受信し，*send*() 関数により応答メッセージを送信する．

実行結果，考察

図 1 にサーバ側の実行結果を示す．受信した要求メッセージまでは表示することができている．また，図 2 にクライアント側の実行結果を示す．受信した応答メッセージが表示されたが，その後の応答を受信できなかった．

```
gotouyuushinoMacBook-puro:第12回 yushigoto$ ./web-server
listening...
GET /index.html; HTTP/1.1
Host: 16777343

HTTP/1.1 200 OK
Content-Lnegth:
Connection: close
```

図 1: サーバ側の実行結果

```
gotouyuushinoMacBook-puro:第12回 yushigoto$ ./web-client http://localhost:50000/
index.html
connected to 16777343
getting /index.html from 16777343

HTTP/1.1 200 OK
Content-L
negth:
Connection: close

Content-Type: text/htmlse

error: reading a response.
```

図 2: クライアント側の実行結果

感想

今回，Python や Java ではなく C 言語を用いることによって，できるだけスクラッチに近い Web クライアントサーバモデルの実装を行った．そのため，より Web クライアントやサーバのアルゴリズムを理解することができた．

参考文献

[*C* 言語] 手入力で返す HTTP サーバ，Qiia（$https://qiita.com/edo_m18/items/4ba1837cf9182005370a$）

付録

Listing 1: Web-client.c

```c
#include <sys/socket.h>
#include <netdb.h>
#define PORT 50000

int main(int argc, char *argv[]){
  int sockfd; // ソケットファイルディスクリプタ
  struct sockaddr_in serv; // アドレス情報構造体
  struct hostent *host;
```

```
9   char buff[1024]; // 読み書き用バッファ
10
11  if ((sockfd = socket(PF_INET, SOCK_STREAM, 0)) < 0 ){
12    perror("socket() failed.");
13    exit(1);
14  }
15
16  char host0[1024], path[1024], *p;
17  int port;
18  sscanf(argv[1], "http://%s", host0);
19  p = strchr(host0, '/');
20  strcpy(path, p);
21  *p = '\0';
22  p = strchr(host0, ':');
23  port = atoi(p+1);
24  *p = '\0';
25
26  if ((host = gethostbyname(host0)) == "NULL"){
27    fprintf("host not foutd\n");
28    exit(1);
29  }
30  bzero((void *)&serv, sizeof(serv));
31  bcopy(host->h_addr_list[0], &serv.sin_addr, host->h_length);
32  serv.sin_family = AF_INET;
33  serv.sin_port = htons(port); // ホストの任意アドレスに設定
34
35  if (connect(sockfd, (struct sockaddr *) &serv, sizeof(serv)) < 0){
36    fprintf("connect error\n");
37    exit(1);
38  }
39  printf("connected to %d\n", *(unsigned int *)host->h_addr_list[0]);
40  printf("getting %s from %d\n\n", path, *(unsigned int *)host->
        h_addr_list[0]);
41
42  sprintf(buff, "%d %d %s", *(unsigned int *)host->h_addr_list[0], port
        , path);
43  send(sockfd, buff, strlen(buff), 0);
44
45  // scanf("%s\n", buff);
46  // send(sockfd, buff, strlen(buff), 0);
47
48  while (1){
49    // char buff0[1024];
50    if (recv(sockfd, buff, strlen(buff), 0) <= 0){
51      printf("error: reading a response.\n");
52      exit(1);
53    }
54    printf("%s\n", buff);
55  }
```

```
56 }
```

Listing 2: Web-server.c

```
1  #include <sys/socket.h>
2  #include <unistd.h>
3  #include <netinet/in.h>
4  #include <arpa/inet.h> // バイトオーダ変換用
5  #include <strings.h> // bzero()用
6  #define PORT 50000
7
8  int main(){
9    int sockfd; // 待受用ソケット（ソケットファイルディスクリプタ）
10   int connectfd; // 通信用ソケット
11   struct sockaddr_in serv; // アドレス情報構造体
12   struct sockaddr_in addr; // 接続相手（クライアント）のアドレス情報格
         納用
13   socklen_t addrlen;
14   // pid_t childpid; // 用 fork
15   char buff[1024];
16
17   if ((sockfd = socket(PF_INET, SOCK_STREAM, 0)) < 0 ){
18     perror("socket() failed.");
19     exit(1);
20   }
21
22   bzero((void *)&serv, sizeof(serv));
23   serv.sin_family = AF_INET;
24   serv.sin_addr.s_addr = htonl(INADDR_ANY);
25   serv.sin_port = htons(PORT); // ホストの任意アドレスに設定
26
27   if (bind(sockfd, (struct sockaddr *) &serv, sizeof(serv)) < 0){
28     fprintf("bind error\n");
29     exit(1);
30   }
31   listen(sockfd, 10);
32   printf("listening...\n");
33
34   char buff0[1024];
35   char buff1[1024];
36   char host[1024], port[1024], path[1024];
37   addrlen = sizeof(addr);
38   if ((connectfd = accept(sockfd, (struct sockaddr *) &addr, &addrlen))
         < 0){
39     fprintf("accept error\n");
40     exit(1);
41   }
42   recv(connectfd, buff0, sizeof(buff0), 0); // ソケットから受信
43   sscanf(buff0, "%s %s %s", host, port, path);
```

```
44
45   printf("GET %s HTTP/1.1\n", path);
46   printf("Host: %s\n\n", host);
47
48   sprintf(buff1, "HTTP/1.1 200 OK \nContent-Lnegth: \nConnection: close
         \nContent-Type: text/html");
49   printf("%s\n", buff1);
50   send(connectfd, buff1, strlen(buff1), 0); // ソケットへ送信
51 }
```