計算機科学実験及び演習 2 報告書 1
課題 3 その 2

工学部情報学科計算機科学コース 25 年度入学 2 回生　勝見久央　1029-25-5242

提出日: 2014 年 11 月 11 日

1　プログラム仕様

1.1　外部仕様

1.1.1　動作の概要

このプログラムは HTTP サーバを c 言語で実装したものである。クライアントからの HTTP リクエストを受信するとそれに応じてクライアントにコンテンツを送信する。

1.1.2　ファイルの説明・コンパイル方法および実行方法

ソースコードは server.c というファイルに記述されている。実行時はターミナルから実行ファイル名とポート番号を以下のように指定して実行する。以下の例ではソースファイル server.c をコンパイルして server という実行ファイルを作成し、実行時はその実行ファイル名とポート番号として 30001 を指定して実行している。

```
katsumihisao-no-MacBook-Air% ./server 30001
```

1.1.3　コマンドライン引数の説明

このプログラムはコマンドライン引数として int 型のポート番号を必要とする。

1.1.4　操作方法

サーバ側では特に操作は必要ない。

1.1.5　入力データの形式

上で述べたコマンドライン引数以外には入力データは必要ない。

1.1.6　出力データの形式，等々

データ出力はない。

1.2 内部仕様

1.2.1 モジュール構成

モジュールはなし。

1.2.2 (自分で定義した) データ構造 (構造体・共用体) の説明

なし。.

1.2.3 大域変数の名前、意味、参照関係の説明

大域変数のうち特に説明が必要なものについて以下で説明する。

- int s
  ソケットの生成に成功した時に socket() から返される識別子。
- int ns
  受け付けたソケットに与えられた識別子を返り値とする accept() から返される値。
- FILE *fp
  クライアントからのリクエストを指すファイルポインタ。
- FILE *fp2
  クライアントから指定されたファイルを指すファイルポインタ。
- int pid
  プロセス分岐時に各プロセスに与えられる ID を格納する変数。ただし子プロセス作成に失敗した場合は-1 になり、子プロセスではこの値は 0 になる。

1.3 実行の手順

同一計算機上で作成したサーバの動作を確認するため以下の様な手順をとった。なおクライアントとして用いたのは前回作成したクライアントでありソースコードは client.c に記載されている。これは server.c と同一ディレクトリに置かれている。また、そのディレクトリ内に test/test.txt となるようにファイルを新しく作成する。test.txt 内は適当な文字列でよい。今回は以下の様な内容にした。

```
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa

bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb

ccccccccccccccccccccccccccccccccccc

1111111111111111111111111111111111

2222222222222222222222222222222222
```

```
33333333333333333333333333333333

44444444444444444444444444444444

55555555555555555555555555555555
```

この状態でターミナルを全部で 4 つ開く。これらのターミナルをそれぞれ便宜上 se、cl1、cl2、cl3 と呼ぶことにする。以下でそれぞれのターミナルで行った手順を説明する。なおこの動作確認時は server.c 内においてコメントアウトされているマクロ「DEBUG」を有効にする。まずターミナル se では make コマンドで必要なファイルを全てコンパイルした後、サーバを起動させる。手順は以下の通り。

```
katsumihisao-no-MacBook-Air% make
gcc -Wall -g -c server.c
gcc -o server server.o
gcc -Wall -g -c client.c
gcc -o client client.o
katsumihisao-no-MacBook-Air% ./server 30001
```

これでサーバはクライアントからのリクエストを待機している状態になる。
次に、ターミナル cl1 からは server.c と同一ディレクトリ内にクライアント自身のソースファイルである client.c というファイルを表示するリクエストをサーバに送信する。手順は以下の通り。ただしプロンプトに入力した最後の行（3 行目）を入力後も改行は入力しないでおく。なおこのことは残りの 2 つのターミナル cl2、cl3 についても同様である。

```
katsumihisao-no-MacBook-Air% ./client
Enter the URL>>http://localhost:30001/client.c
```

さらに、ターミナル cl2 からは先ほど用意した test.txt の内容を表示するリクエストをサーバに送信する。手順は以下の通り。ただし、このターミナルからはコンパイルは行わずに先ほど作成した実行ファイルを指定して実行することとする。これは cl3 についても同様である。

```
katsumihisao-no-MacBook-Air% ./client
Enter the URL>>http://localhost:30001/test/test.txt
```

最後にターミナル cl3 からは無効なリクエストをサーバに送信する。具体的な手順は以下の通り。

```
katsumihisao-no-MacBook-Air% ./client
Enter the URL>>http://localhost:40000/hogehoge
```

これで全てのクライアントの準備が整ったことになり、これらを順に実行していく。具体的には cl1 から cl3 まで順にプロンプトに入力したままになったアドレスの後に改行を入力していく。

1.4 出力結果

上記のような手順でプログラムを実行すると 3 つのクライアントの処理をサーバが同時に行うことができていることが確認できる。（プログラム中の sleep(1) を有効にしたのはこれをはっきり

と確認するためである。）処理は同時に始まり、おそらく cl3、cl2、cl1 の順に終わる。cl1 から cl3 のそれぞれの出力は順に以下のようになった。以下は cl1 の出力結果

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=us-ascii
#include<sys/socket.h>
#include<netinet/in.h>
#include<netdb.h>
#include<unistd.h>
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<errno.h>

#define NUMSTR  2

char *reqlines[NUMSTR] = {
    "Request from client\r\n",
    "\r\n"
};

Number);

#ifdef DEBUG
        printf("Completed anlyzing the URL\n");
        printf("Host Name = %s \n",HostName);
        printf("Port Number = %s \n",PortNumber);
        printf("Path = %s \n",Path);
#endif


~中略~


        //?^^ea^^a5^^af???????^^c8^^a4??????????^^eb^^a1^^a3
        for(i = 0; i < NUMSTR; i++){
            send(s, reqlines[i], strlen(reqlines[i]), 0);
        }

        // receive contents from server
        while (fgets(buf, sizeof(buf), fp) != NULL){
            printf("%s", buf);
        }
```

```
        close(s);
    }
    return 0;
}
```

```
Enter the URL>>
```

以下は cl2 の出力結果。

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=us-ascii
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa

bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb

ccccccccccccccccccccccccccccccccc

1111111111111111111111111111111111

2222222222222222222222222222222222

3333333333333333333333333333333333

4444444444444444444444444444444444

5555555555555555555555555555555555
Enter the URL>>
```

以下は cl3 の出力結果。

```
client connect(): Connection refused
```

これらの結果から、サーバがクライアントからリクエストを受信するたびに新しくプロセスを作成してその子プロセスで処理を行うという設計をとっているため、ひとつのクライアントからの処理が終了するのを待たなくても他のクライアントから新しくリクエストを受信してもすぐに応答することができたり、サーバが現在応答中のプロセスが異常終了しても他のクライアントからのリクエストに影響をおよぼすことなくそれらに対しては正常な動作を行うことができるということが確認できた。

1.5 採用した設計について

複数クライアントからの処理に対応するために fork() による分岐を行う設計を採用した。

2　感想

今回は 2 回の課題にわたってサーバを作成したのだが、前回のクライアント作成時と同様に初めて c 言語でここまで実用的なプログラムを書くことになりより一層 c 言語の面白さというものを感じられた気がする。またソケットやポートなどネットワーク接続に関する基本的な事項についてもより理解を深めることができ大変有意義な課題であったと思う。

3　参考文献

ハーバート・シルト　著「独習 C 第 4 版」