情報科学演習 C
第 2 回レポート

ネットワークプログラミング

担当教員 : 小島 英春，内山 彰 教員
提出者 : 川上 遼太
所属/学年 : 基礎工学部 情報科学科 計算機科学コース 3 年
学籍番号 : 09B16022
電子メール : u407773h@ecs.osaka-u.ac.jp

提出日 : 2018 年 5 月 27 日 (日)
締切日 : 2018 年 5 月 28 日 (月) 12:59

目 次

1 (課題 2-1) echoclient.c

この章では TCP クライアントプログラムの作成について記述する．配布された TCP サーバープログラムと作成した TCP クライアントプログラムは A.1，A.2 節に記載している．

1.1 課題内容

配布された echoserver プログラムと通信することのできるクライアントプログラムを作成せよ．

1.2 仕様

echoclient プログラムの仕様は以下の通りである．

- 実行のコマンドは”echoclient host”と入力する．host の部分には echoserver プログラムを実行しているホスト名を指定すること．
- echoserver プログラムと echoclient プログラムはポート番号 10120 を介して接続する．
- echoclient プログラムの標準入力から読み込んだ文字列を echoserver プログラムに送信し，echoserver プログラムが受け取った文字列を再度 echoclient プログラムに送信し，echoclient プログラムの標準出力に出力する．送信する文字列は改行までであり，入力の終わりは EOF を読み込んだ時である．
- 接続が切れた時，echoserver プログラム側で”closed”と出力する．
- echoserver プログラムは echoclient プログラムとの接続が一旦終了しても次に接続されるまで起動し続ける．
- 入力できる文字列の長さは半角で 1023 文字である．

1.3 実行結果

以下に実行結果を記載する．

ssh によるリモートログイン

```
1 r-kawakm@exp048:~$ ssh exp068
2 The authenticity of host 'exp068 (192.168.16.47)' can't be established.
3 ECDSA key fingerprint is
   SHA256:7b17ZyM1wvrsUa5xGcYXoFe18QYkzcrS44oSk02MOAg.
4 Are you sure you want to continue connecting (yes/no)? yes
5 Warning: Permanently added 'exp068,192.168.16.47' (ECDSA) to the list of
   known hosts.
6 r-kawakm@exp068's password:
7 Welcome to Ubuntu 16.04.4 LTS (GNU/Linux 4.4.0-124-generic x86_64)
8
```

```
 9  * Documentation:  https://help.ubuntu.com
10  * Management:     https://landscape.canonical.com
11  * Support:        https://ubuntu.com/advantage
12
13 0 個のパッケージがアップデート可能です。
14 0 個のアップデートはセキュリティアップデートです。
```

ターミナルのウィンドウを 2 つ開いて，片方のウィンドウで ssh コマンドを用いて，exp068 というホストにリモートログインした．2 つのターミナルのウィンドウが別々のホストに接続されていることによって，echoserver プログラムと echoclient プログラムが端末内でメッセージのやり取りをしているのではなく，ネットワークを通じてメッセージをやり取りしているとわかる．表 1 に echoserver プログラムと echoclient プログラムのホストの割り当てについてまとめてある．

プログラム	echoserver プログラム	echoclient プログラム
ホスト名	exp048	exp068

表 1: (課題 2-1) における echoserver プログラムと echoclient プログラムのホスト名

echoserver プログラム実行結果

```
1 r-kawakm@exp048:~/enshuC/enshuC_kadai2$ ./echoserver.exe
2 [exp068.exp.ics.es.osaka-u.ac.jp]
3  closed
```

echoserver プログラムを実行しておき，2 行目でホスト exp068 との接続が確立されたことがわかった．3 行目では通信相手である echoclient プログラムが通信を終了したことが”closed”と出力されていることによって確認できた．

echoclient プログラム実行結果

```
1 r-kawakm@exp068:~/enshuC/enshuC_kadai2$ ./echoclient.exe exp048
2 echoclient->
3 echoclient->
4  Hello World!
5  Hello World!
```

ホスト exp048 で echoserver プログラムを実行している状態で，先ほど ssh コマンドによってホスト exp068 にリモートログインしたターミナルのウィンドウ上で echoclient プログラムを実行し，echoserver プログラム側 のウィンドウで”[exp068.exp.ics.es.osaka-u.ac.jp]”という出力があることで，接続されたことが確認できる．配布された手引きでは”echclient host”で実行するよう指示されているが，コンパイル時に生成ファイル名を echoclient に設定すればそのようになり，ファイル名の違いなので問題ないと考えている．

2，4 行目の文字列が入力した文字列であり，3，5 行目でおうむ返しのように入力した文字列と全く同じ文字列が出力されていることが確認でき，”Ctrl+D”で echoclient プログラムが終了したため，1.2 節で記載した仕様 (文字列の長さ以外) を満たしていることが確認できた．文字列の長さ

についての仕様の確認は行っていないが，文字列を保存する配列の長さが 1024 であり，文字列の終端記号の分を除いた 1023 個を入力文字列に当てることができることから仕様が満たされていることを確認できる．

空白を含んだ文字列でも問題なく処理できていることに注意して，1.4 節に移ってほしい．

1.4 プログラム説明

この節では echoclient プログラムの説明を行う．echoserver プログラムは配布された資料のものを実行しているため，本レポートでは説明しない．はじめにも伝えたが，echoserver プログラムと echoclient プログラムの全行のコードは A.1，A.2 節に記載している．

ポート番号の定義

```
10 #define PORT 10120
```

10 行目でポート番号を定義していて，echoserver プログラムと echoclient プログラムは 10120 番を使用する．

変数の宣言

```
13   int sock;
14   char str[1024], receive[1024];
15   struct sockaddr_in host;
16   struct hostent *hp;
```

13-16 行で本プログラムで使用する変数を宣言している．変数の型と名前，用途を表 2 にまとめる．

型	変数名	用途
int 型	sock	ソケットの生成に用いる
char 型	str[1024]	入力された文字列を保存する
char 型	receive[1024]	echoserver プログラムから受信した文字列を保存する
構造体 sockaddr_in 型	host	接続先のソケットアドレスを保存する
構造体 hostent 型	*hp	echoserver プログラムのホスト情報を保存する

表 2: echoclient プログラムで使用する変数

引数の数の確認

```
18   /* コマンド引数が 1 つであることを確認 */
19   if(argc != 2) {
20     fprintf(stderr, "usage: %s host\n", argv[0]);
21     exit(1);
22   }
```

コマンド引数が 2 つであることを確認し，コマンド引数が 2 つでない時は間違った実行方法であるため，正しい実行方法をエラー出力して，echoclient プログラムを終了する．

ソケットの生成

```
24   /* ソケットの生成 */
25   if((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
26     perror("client: socket");
27     exit(1);
28   }
```

socket 関数を用いることで通信の出入り口となるソケットを生成する．また，socket 関数の返り値が 0 以下であれば，正しく生成できていないためエラー出力をして，echoclient プログラムを終了する．

接続先ホストの設定，ホスト情報の保存

```
30   /*
31    * host 構造体に接続するサーバーのアドレス・ポート番号を設定
32    */
33
34   bzero((char *)&host, sizeof(host));
35   host.sin_family = AF_INET;
36   host.sin_port = htons(PORT);
37
38   if((hp = gethostbyname(argv[1])) == NULL) {
39     fprintf(stderr, "unkown host %s\n", argv[1]);
40     close(sock);
41     exit(1);
42   }
43   bcopy(hp->h_addr, &host.sin_addr, hp->h_length);
```

接続先ホストの設定を変数 host に反映させて，接続する準備を行う．設定を行う前に bzero 関数を用いて変数の初期化を忘れないようにすること．

その後，実行時に引数として与えられたホスト名のホストに接続を試み，成功すればホスト情報をポインタ変数*hp に bcopy 関数を用いて保存し，失敗すればエラー出力をしてソケットを閉じた後，echoclient プログラムを終了する．

ソケットの接続

```
45   /* ソケットをサーバに接続 */
46   if (connect(sock, (struct sockaddr *)&host, sizeof(host)) > 0) {
47     perror("client: connect");
48     close(sock);
49     exit(1);
```

```
50   }
```

connect 関数を用いて生成したソケットを echoserver プログラムのソケットと接続する．接続に失敗した時にはエラー出力をしてソケットを閉じた後，echoclient プログラムを終了する．

入力，送受信，出力

```
52 memset(str, '\0', sizeof(str));
53
54 while (read(0, str, sizeof(str)) > 0) {
55   memset(receive, '\0', sizeof(receive));
56   if(write(sock, str, sizeof(str)) < 0) {
57     perror("write");
58     exit(1);
59   }
60
61   while (read(sock, receive, sizeof(receive)) < 0) {
62     perror("read");
63   }
64     write(1, receive, sizeof(receive));
65   // printf("%s\n", receive);
66 }
67
68   /* ソケットをクローズ */
69   close(sock);
70 }
```

はじめに入力文字列を保存する変数 str，receive を memset 関数を用いることで初期化する．初期化することで，文字化けのようなゴミ文字列を出力しないようにする．

まず，read 関数で標準入力から文字列を受け取り，変数 str に格納する．その後 write 関数を用いてソケットを介したデータ送出を行う．送信した後，サーバーからデータを受信するまで read 関数を用いて，受け取った後 write 関数を用いて echoclient プログラム側の標準出力に受け取った文字列を出力する．そして，次の入力を受け付ける．この処理を EOF(End Of File) を受け取るまで行い，EOF を受け取ったら，ソケットを閉じれ echoclient プログラムの終了となる．

はじめ，標準入力からの文字列の受け取りに scanf 関数を用いて，出力に printf 関数を用いていたが，この時は空白を含んだ文字列が正しく処理することができず，1 行で出力されないといけないところが，下の例のように空白の部分で改行されて文字列が出力されていた．なぜ，そのような結果になったか解明できていないが，そのような結果になることを避けるために write 関数と read 関数を用いてプログラムを記述した．

例

```
入力 : Hello World!
出力 : Hello
       World!
```

2 (課題 2-2) simple-talk プログラム

この章では簡易 talk プログラムについて記述する．簡易 talk プログラムは echoserver プログラムと echoclient プログラムのように echoclient プログラムが送信した内容がおうむ返しで echoclient プログラムに帰ってくると言うブーメランのような機能ではなくて，simple_talk_server プログラムと simple_talk_client プログラムが各々入力した文字列が相手のウィンドウにリアルタイムで出力されるというチャットのような機能を有しているプログラムである．

また，作成した simple_talk_server プログラムと simple_talk_client プログラムのコードを A.3, A.4 節に記載している．

2.1 課題内容

本課題の配布資料である手引きを参考に simple_talk_server プログラムと simple_talk_client プログラムを作成せよ．

2.2 仕様

simple_talk_server プログラムと simple_talk_client プログラムの仕様は以下の通りである．

- 実行のコマンドは”echoclient host”と入力する．host の部分には simple_talk_server プログラムを実行しているホスト名を指定すること．
- simple_talk_server プログラムと simple_talk_client プログラムはポート番号 10130 を介して接続する．
- simple_talk_server プログラムの標準入力に入力された文字列はリアルタイムで simple_talk_client プログラムに送信されて simple_talk_client プログラムの標準出力に出力される．逆に simple_talk_client プログラムの標準入力に入力された文字列も同様にリアルタイムで simple_talk_server プログラムに送信されて simple_talk_server プログラムの標準出力に出力される．
- simple_talk_server プログラムと simple_talk_client プログラムの接続が確立した時，両プログラムの標準出力に”connected”と出力することによって，両プログラムのユーザーに接続の確立を伝える．
- simple_talk_server プログラムと simple_talk_client プログラムは”Ctrl + D”で EOF(End oF File) がどちらかに入力された時に接続を終了する．
- 接続が切れた時, simple_talk_server プログラムと simple_talk_client プログラムの両方で”closed”と標準出力に出力する．
- simple_talk_server プログラムは simple_talk_client プログラムとの接続が一旦終了しても次の接続されるまで起動し続ける．
- 入力できる文字列の長さは半角で 1023 文字である．

2.3 実行結果

ssh コマンドを用いたリモートログインの実行結果は 1.3 節の”ssh によるリモートログイン”と同じため，省略する．

simple_talk_server

```
1 r-kawakm@exp048:~/enshuC$ ./simple_talk_server.exe
2 [exp068.exp.ics.es.osaka-u.ac.jp]
3 connected
4 cl->sv
5 sv->cl
6 closed
7
```

配布された手引きでは”simple-talk-server”で実行するよう指示されているが，コンパイル時に生成ファイル名をに”simple-talk-server”設定すればそのようになり，ファイル名の違いなので問題ないと考えている．

2，3 行目がホスト exp068 の simple_talk_client プログラムとの接続が確立したことをユーザーに知らせる出力である．”cl-¿sv”は simple_talk_client プログラムから送信されたデータを simple_talk_server プログラムで受け取り，出力している例であり，”sv-¿cl”は simple_talk_server プログラムから送信されたデータを simple_talk_client プログラムで受け取り，出力している例である．ここでは simple_talk_client プログラムから送られてきた情報を受信して，simple_talk_server プログラムで出力していることを確認しておく．

6 行目では”closed”と出力されていることから通信が終了したことは確認できる．その後次の接続の待機に移行している．

simple_talk_client

```
1 r-kawakm@exp068:~/enshuC$ ./simple_talk_client.exe exp048
2 connected
3 cl->sv
4 sv->cl
5 closed
```

1 行目で仕様の通りにホスト名を引数として与えて実行している．配布された手引きでは”simple-talk-client host”で実行するよう指示されているが，コンパイル時に生成ファイル名をに”simple-talk-client”設定すればそのようになり，ファイル名の違いなので問題ないと考えている．また 2 行目において”connected”と出力されていることから simple_talk_server プログラムとの接続が確立したことが確認できる．3，4 行目については simple_talk_server プログラムの出力結果のところで記した通りであり，ここでは simple_talk_server プログラムから送られてきた情報を受信して，simple_talk_client プログラムで出力していることを確認しておく．6 行目では”closed”と出力されていることから通信が終了したことは確認できる．接続の終了は simple_talk_server プログラム，simple_talk_client プログラムのどちらの標準入力で EOF が入力されたときに行われる．

文字列の長さについての仕様の確認は行っていないが，文字列を保存する配列の長さが 1024 で

あり，文字列の終端記号の分を除いた 1023 個を入力文字列に当てることができることから仕様が満たされていることを確認できる．

simple_talk_server プログラムと simple_talk_client プログラムの出力結果から送受信がうまく行われていることが確認できた．しかし，次のような場合には少し変な挙動を起こす．ただし，今回の課題で求められていることは満たしているため問題はないと考えている．

入力中にデータを受信した時:受信側

```
1 r-kawakm@exp048:~/enshuC$ ./simple_talk_server.exe
2 [exp068.exp.ics.es.osaka-u.ac.jp]
3 connected
4 sv->cl->sv
5
```

入力中にデータを受信した時:送信側

```
1 r-kawakm@exp068:~/enshuC$ ./simple_talk_client.exe exp048
2 connected
3 cl->sv
```

この出力の際の状況としては，simple_talk_server プログラムの標準入力に”sv-¿”と入力している途中で，simple_talk_client プログラムの標準入力に”cl-¿sv”と入力して送信したところ，simple_talk_server プログラムへの出力が入力途中であった”sv-¿”に続いて”sv-¿cl-¿sv”となっているのである．これは simple_talk_client プログラムが入力中で simple_talk_server プログラムが送信してきても同じである．入力の途中にデータを受信した際に改行して出力することも考えたが，そうすると入力を続けることが困難になると考え，さらに課題の最低限の仕様を満たしていると考えたためにこのままにしている．

LINE アプリのように入力フィールドと出力フィールドが分かれていれば良いが，本課題のように 1 つのターミナルウィンドウで入出力を行う場合には改善は難しいのではないかと考えている．対処方法としては入力を受け付けている時にデータを受信しても，入力が終わるまで出力しないというのが考えられるが，これではリアルタイムとは言い難いのではないかと考えている．

2.4 プログラム説明

この節では simple_talk_server プログラムと simple_talk_client プログラムの説明を行う．

2.4.1 simple_talk_server プログラムの説明

まず，simple_talk_server プログラムについて説明する．

ポート番号の定義

```
9 #define PORT 10130
```

9 行目でポート番号を定義している．simple_talk_server プログラムと simple_talk_client プログラムは 10130 番を使用する．

変数の宣言

```
12   int sock, csock; /* クライアントを受け付けたソケット*/
13   struct sockaddr_in svr;
14   struct sockaddr_in clt;
15   struct hostent *cp;
16   int clen;
17   char sbuf[1024], cbuf[1024];
18   int nbytes;
19   int reuse;
20   fd_set rfds; /* select() で用いるファイル記述子集合*/
21   struct timeval tv; /* select() が返ってくるまでの待ち時間を指定する変数*/
```

以下の表 3 に simple_talk_server プログラムで使用する変数の型と名前，用途をまとめる．

型	変数名	用途
int 型	sock	送信用ソケットの生成に用いる
int 型	csock	受信用ソケットの生成に用いる
構造体 sockaddr_in 型	svr	受付用のソケットアドレスを保存する
構造体 sockaddr_in 型	clt	接続先のソケットアドレスを保存する
構造体 hostent 型	*cp	simple_talk_client プログラムのホスト情報を保存する
int 型	clen	変数 clt のサイズを保存する
char 型	sbuf[1024]	入力された文字列を保存する
char 型	cbuf[1024]	simple_talk_client プログラムから受信した文字列を保存する
int 型	nbytes	read 関数の返り値を保存する．
int 型	reuse	ソケットの再利用に使用する
fd_set 型	rfds	select 関数で用いるファイル記述子の集合
構造体 timeval 型	tv	select 関数が帰ってくるまでの待ち時間を指定する

表 3: simple_talk_server プログラムで使用する変数

ソケットの生成

```
23   /* ソケットの生成 */
24   if ((sock = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP)) < 0) {
25     perror("socket");
26     exit(1);
27   }
```

socket 関数を用いることで通信の出入り口となるソケットを生成する．また，socket 関数の返り値が 0 以下であれば，正しく生成できていないためエラー出力をして，simple_talk_server プログ

ラムを終了する．

ソケットアドレス再利用の指定

```
29   /* ソケットアドレス再利用の指定 */
30   reuse = 1;
31   if(setsockopt(sock, SOL_SOCKET, SO_REUSEADDR, &reuse, sizeof(reuse))
       < 0) {
32     perror("setsockopt");
33     exit(1);
34   }
```

ここではソケットアドレスの再利用をするための処理を行っている．

クライアントを受け付けるためのソケット情報の設定

```
36   /* client 受付用ソケットの情報設定 */
37   bzero(&svr, sizeof(svr));
38   svr.sin_family = AF_INET;
39   svr.sin_addr.s_addr = htonl(INADDR_ANY);
40
41   /* 受付側の IP アドレスは任意 */
42   svr.sin_port = htons(PORT); /* ポート番号 10130 を介して受け付ける */
43
44   /* ソケットにソケットアドレスを割り当てる */
45   if(bind(sock, (struct sockaddr *)&svr, sizeof(svr)) < 0) {
46     perror("bind");
47     exit(1);
48   }
```

simple_talk_client プログラムを受け付けるためのソケットの情報を設定している．設定する前の bzero 関数を用いて変数を svr を初期化している．設定した後，ポート番号を 10130 番に設定している．そして，bind 関数を用いて変数 sock と変数 svr を結びつけている．この時結びつけに失敗した時にはエラー出力をして simple_talk_server プログラムを終了する．

待ち受けクライアント数の設定

```
50   /* 待ち受けクライアント数の設定 */
51   if(listen(sock, 5) < 0) { /* 待ち受け数に 5 を指定 */
52     perror("listen");
53     exit(1);
54   }
```

listen 関数を用いてクライアントの受付数を 5 に設定している．指定できなかった時にはエラー出力をして simple_talk_server プログラムを終了する．

受付，入力，受信，出力，終了

```
55   do {
56
57     /* クライアントの受付 */
58     clen = sizeof(clt);
59     if ((csock = accept(sock, (struct sockaddr *)&clt, &clen)) < 0) {
60       perror("accept");
61       exit(2);
62     }
63
64     /* クライアントのホスト情報の取得 */
65     cp = gethostbyaddr((char *)&clt.sin_addr, sizeof(struct in_addr),
        AF_INET);
66     printf("[%s]\n", cp->h_name);
67     printf("connected\n");
68
69     nbytes = -1;
70
71     do{
72       /* 入力を監視するファイル記述子の集合を変数 rfds にセットする*/
73       FD_ZERO(&rfds); /* rfds を空集合に初期化*/
74       FD_SET(0, &rfds); /* 標準入力*/
75       FD_SET(csock, &rfds); /* クライアントを受け付けたソケット*/
76       /* 監視する待ち時間を 1 秒に設定*/
77       tv.tv_sec = 1;
78       tv.tv_usec = 0;
79       memset(sbuf, '\0', sizeof(sbuf));
80       memset(cbuf, '\0', sizeof(cbuf));
81
82       /* 標準入力とソケットからの受信を同時に監視する */
83       if(select(csock+1, &rfds, NULL, NULL, &tv) > 0) {
84         if(FD_ISSET(0, &rfds)) { /* 標準入力から入力があったなら*/
85           if((nbytes = read(0, sbuf, sizeof(sbuf))) > 0){
              /*標準入力から読み込みクライアントに送信 */
86             write(csock, sbuf, nbytes);
87           }
88         }
89         if(FD_ISSET(csock, &rfds)) { /* ソケットから受信したなら*/
90         /* ソケットから読み込み端末に出力*/
91          if((nbytes = read(csock, cbuf, sizeof(cbuf))) > 0){
92             write(1, cbuf, nbytes);
93          }
```

```
94          }
95        }
96      } while (nbytes != 0);
97
98      /* read() が 0 を返すまで (=End-Of-File) 繰り返す */
99      close(csock);
100      printf("closed\n");
101    } while(1); /* 繰り返す*/
```

57-67 行目ではクライアントを受け付けて，クライアントのホスト情報を取得する．73-78 行目では入力を監視するファイル記述子の集合を変数 rfds に保存する．73 行目で変数 rfds を FD_ZERO マクロを用いて初期化する．74，75 行目で FD_SET マクロの第 1 引数に 0，csock を与えることで標準入力と simple_talk_client プログラムからの受信を代入している．また，77-78 行目で入力を監視する待ち時間を 1 秒に設定している．79-80 行目では標準入力とソケットから受信した文字列を保存する変数 sbuf，cbuf の初期化を memset 関数を用いて行っている．83-96 行目では select 関数を用いて標準入力とソケットからの受信を同時に監視している．標準入力から入力があった時には read 関数で変数 sbuf にその入力を保存して，write 関数で simple_talk_client プログラムに送信する．ソケットから受信したなら受信した文字列を変数 cbuf にその文字列を保存して，write 関数で標準出力にその文字列を出力する．この監視の処理を EOF(End Of File) を受け付けるまで繰り返す．EOF を受け付けたら接続されている simple_talk_client プログラムのソケットを閉じて，接続が終了したことを”closed”と標準出力に出力することでユーザーに知らせる．そして他のクライアントプログラムからのソケット受信を待ち続ける．

2.4.2 simple_talk_client プログラムの説明

ここでは simple_talk_client プログラムの説明を行う．

ポート番号の定義

```
9 #define PORT 10130
```

9 行目でポート番号を定義している．simple_talk_server プログラムと simple_talk_client プログラムは 10130 番を使用する．

変数の宣言

```
12 int sock, nbytes;
13 char sbuf[1024], *name, cbuf[1024];
14 struct sockaddr_in svr;
15 struct hostent *hp;
16 fd_set rfds;
17 struct timeval tv;
```

以下の表 4 に simple_talk_client プログラムで使用する変数の型と名前，用途をまとめる．

型	変数名	用途
int 型	sock	送信用ソケットの生成に用いる
int 型	nbytes	read 関数の返り値を保存する.
char 型	*name	
char 型	sbuf[1024]	simple_talk_server プログラムから受信した文字列を保存する
char 型	cbuf[1024]	入力された文字列を保存する
構造体 sockaddr_in 型	svr	受付用のソケットアドレスを保存する
構造体 hostent 型	*hp	simple_talk_server プログラムのホスト情報を保存する
fd_set 型	rfds	select 関数で用いるファイル記述子の集合
構造体 timeval 型	tv	select 関数が帰ってくるまでの待ち時間を指定する

表 4: simple_talk_client プログラムで使用する変数

引数の数の確認

```
19 if(argc != 2) {
20 fprintf(stderr, "Usage: %s hostname \n", argv[0]);
21 exit(1);
22 }else{
23   name = *(argv+1);
24 }
```

コマンド引数が 2 つであることを確認し，コマンド引数が 2 つでない時は間違った実行方法であるため，正しい実行方法をエラー出力して，simple_talk_client プログラムを終了する.

ソケットの生成

```
26 if((sock = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP)) < 0){
27   perror("socket");
28   exit(1);
29 }
```

socket 関数を用いることで通信の出入り口となるソケットを生成する．また，socket 関数の返り値が 0 以下であれば，正しく生成できていないためエラー出力をして，simple_talk_client プログラムを終了する.

接続先ホストの設定，ホスト情報の保存

```
31 bzero(&svr, sizeof(svr));
32 svr.sin_family = AF_INET;
33 svr.sin_port = htons(PORT);
34 if((hp = gethostbyname(name)) == NULL){
35   perror("gethostbyname");
36   exit(1);
```

```
37 }
38 bcopy(hp->h_addr, &svr.sin_addr, hp->h_length);
```

接続先ホストの設定を変数 host に反映させて，接続する準備を行う．設定を行う前に bzero 関数を用いて変数の初期化を忘れないようにすること．

その後，実行時に引数として与えられたホスト名のホストに接続を試み，成功すればホスト情報をポインタ変数*hp に bcopy 関数を用いて保存し，失敗すればエラー出力をしてソケットを閉じた後，simple_talk_client プログラムを終了する．

ソケットの接続

```
40 if((connect(sock, (struct sockaddr *)&svr, sizeof(svr))) < 0){
41     perror("connect");
42     exit(1);
43 }
44 else {
45 printf("connected\n" );
46 }
```

connect 関数を用いて生成したソケットを simple_talk_client プログラムのソケットと接続する．接続に失敗した時にはエラー出力をしてソケットを閉じた後，simple_talk_client プログラムを終了する．接続に成功した時はユーザーに接続が確立したことを伝えるために標準出力に”connected”と出力する．

入力，受信，出力，終了

```
48 do{
49   FD_ZERO(&rfds);
50   FD_SET(0, &rfds);
51   FD_SET(sock, &rfds);
52   tv.tv_sec = 1;
53   tv.tv_usec = 0;
54   memset(sbuf, '\0', sizeof(sbuf));
55 memset(cbuf, '\0', sizeof(cbuf));
56
57   if(select(sock+1, &rfds, NULL, NULL, &tv) > 0){
58   if(FD_ISSET(sock, &rfds)){
59     if((nbytes = read(sock, cbuf, sizeof(cbuf))) > 0){
60       write(1, cbuf, nbytes);
61     }
62   }
63   if(FD_ISSET(0, &rfds)){
64     if((nbytes = read(0, sbuf, sizeof(sbuf))) > 0){
65       write(sock, sbuf, nbytes);
```

```
66     }
67   }
68   }
69 } while(nbytes != 0);
```

49-53 行目では入力を監視するファイル記述子の集合を変数 rfds に保存する．49 行目で変数 rfds を FD_ZERO マクロを用いて初期化する．50，51 行目で FD_SET マクロの第 1 引数に 0，sock を与えることで標準入力と simple_talk_server プログラムからの受信を代入している．また，52-53 行目で入力を監視する待ち時間を 1 秒に設定している．54-55 行目では標準入力とソケットから受信した文字列を保存する変数 sbuf，cbuf の初期化を memset 関数を用いて行っている．57-68 行目では select 関数を用いて標準入力とソケットからの受信を同時に監視している．標準入力から入力があった時には read 関数で変数 sbuf にその入力を保存して，write 関数で simple_talk_client プログラムに送信する．ソケットから受信したなら受信した文字列を変数 cbuf にその文字列を保存して，write 関数で標準出力にその文字列を出力する．この監視の処理を EOF(End Of File) を受け付けるまで繰り返す．

終了処理

```
70 close(sock);
71 printf("closed\n");
```

EOF を受け付けたら接続されている simple_talk_server プログラムのソケットを閉じて，接続が終了したことを”closed”と標準出力に出力することでユーザーに知らせる．そして simple_talk_client プログラムを終了する．

3 考察

本課題で作成した簡易 talk プログラムには 2.3 節でも記したような不便な部分がある．入力の途中でソケットからデータを受信すると入力している文字列に続いて受け取ったデータを出力するというのは少し問題であるが，該当箇所でも記したように，1 つのターミナルウィンドウで入出力を行っていて，かつリアルタイムで反映させるならば仕方がないことであったと考える．

他には LINE アプリなどの SNS の機能を考えると複数のユーザーとのトークが可能になるような機能を実装できると考えると本当に簡易的なものだと思える．時間があれば，発展問題にも取り組んでさらなる拡張を行いたかったが，残念である．

4 感想

情報ネットワークの講義で初めてネットワークについての勉強を行った状態，つまりネットワークに関する知識が全くない状態で本課題にのぞんだため，とっかかりを探すのに大変困った．ネットワークがそのように接続されて，どのように通信を行っているかも知らなかったため，ソケットについても全くわからずいろいろ調べながら理解して行った．そこそこの知識が揃うと配布された手引きのわかりやすさに感動して，課題を進めることができた．プログラムを組み終えた今さらな

るネットワークの知識を得ることを楽しみにしているため，このモチベーションのまま本講義の終了まで突っ走っていきたいと考えている．

A (付録) プログラムコード

A.1 (課題 2-1) echoserver.c

```
 1  #include <sys/types.h>
 2  #include <sys/socket.h>
 3  #include <netinet/in.h>
 4  #include <netdb.h>
 5  #include <unistd.h>
 6  #include <string.h>
 7  #include <stdio.h>
 8  #include <stdlib.h>
 9
10  int main(int argc,char **argv) {
11    int sock,csock;
12    struct sockaddr_in svr;
13    struct sockaddr_in clt;
14    struct hostent *cp;
15    int clen;
16    char rbuf[1024];
17    int nbytes;
18    int reuse;
19    /* ソケットの生成*/
20    if ((sock=socket(AF_INET,SOCK_STREAM,IPPROTO_TCP))<0) {
21      perror("socket");
22      exit(1);
23    }
24    /* ソケットアドレス再利用の指定*/
25    reuse=1;
26    if(setsockopt(sock,SOL_SOCKET,SO_REUSEADDR,&reuse,sizeof(reuse))<0) {
27      perror("setsockopt");
28      exit(1);
29    }
30    /* client 受付用ソケットの情報設定*/
31    bzero(&svr,sizeof(svr));
32    svr.sin_family=AF_INET;
33    svr.sin_addr.s_addr=htonl(INADDR_ANY);
34    /* 受付側の IP アドレスは任意*/
35    svr.sin_port=htons(10120); /* ポート番号 10120 を介して受け付ける*/
36    /* ソケットにソケットアドレスを割り当てる*/
37    if(bind(sock,(struct sockaddr *)&svr,sizeof(svr))<0) {
38      perror("bind");
39      exit(1);
40    }
41    /* 待ち受けクライアント数の設定*/
42    if (listen(sock,5)<0) { /* 待ち受け数に 5 を指定*/
43      perror("listen");
44      exit(1);
45    }
46    do {
47    /* クライアントの受付*/
48      clen = sizeof(clt);
49      if ( ( csock = accept(sock,(struct sockaddr *)&clt,&clen) ) <0 ) {
```

```
50        perror("accept");
51        exit(2);
52      }
53    /* クライアントのホスト情報の取得*/
54      cp = gethostbyaddr((char *)&clt.sin_addr,sizeof(struct in_addr),AF_INET);
55      printf("[%s]\n",cp->h_name);
56      do {
57        memset(rbuf, '\0', sizeof(rbuf));
58        /* クライアントからのメッセージ受信*/
59        if ( ( nbytes = read(csock,rbuf,sizeof(rbuf)) ) < 0) {
60          perror("read");
61        } else {
62          write(csock,rbuf,nbytes);
63          /* 受信文字列をそのままクライアントへ返す (echo) */
64        }
65      } while ( nbytes != 0 );
66      /* read() が 0 を返すまで (=End-Of-File) 繰り返す*/
67      close(csock);
68      printf("closed\n");
69    }
70    while(1); /* 次の接続要求を繰り返し受け付ける*/
71  }
```

A.2 (課題 2-1) echoclient.c

```
 1  #include <sys/types.h> /* socket() を使うために必要*/
 2  #include <sys/socket.h> /* 同上 */
 3  #include <netinet/in.h> /* INET ドメインのソケットを使うために必要*/
 4  #include <arpa/inet.h> /* 同上 */
 5  #include <netdb.h>
 6  #include <stdio.h>
 7  #include <string.h>
 8  #include <stdlib.h>
 9  #include <unistd.h>
10  #define PORT 10120
11
12  int main(int argc, char **argv) {
13    int sock;
14    char str[1024], receive[1024];
15    struct sockaddr_in host;
16    struct hostent *hp;
17
18    /* コマンド引数が 1 つであることを確認 */
19    if(argc != 2) {
20      fprintf(stderr, "usage: %s host\n", argv[0]);
21      exit(1);
22    }
23
24    /* ソケットの生成 */
25    if((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
26      perror("client: socket");
27      exit(1);
28    }
29
30    /*
31     * host 構造体に接続するサーバーのアドレス・ポート番号を設定
32     */
```

```
33
34    bzero((char *)&host, sizeof(host));
35    host.sin_family = AF_INET;
36    host.sin_port = htons(PORT);
37
38    if((hp = gethostbyname(argv[1])) == NULL) {
39      fprintf(stderr, "unkown host %s\n", argv[1]);
40      close(sock);
41      exit(1);
42    }
43    bcopy(hp->h_addr, &host.sin_addr, hp->h_length);
44
45    /* ソケットをサーバに接続 */
46    if (connect(sock, (struct sockaddr *)&host, sizeof(host)) > 0) {
47      perror("client: connect");
48      close(sock);
49      exit(1);
50    }
51
52  memset(str, '\0', sizeof(str));
53
54  while (read(0, str, sizeof(str)) > 0) {
55    memset(receive, '\0', sizeof(receive));
56    if(write(sock, str, sizeof(str)) < 0) {
57      perror("write");
58      exit(1);
59    }
60
61    while (read(sock, receive, sizeof(receive)) < 0) {
62      perror("read");
63    }
64      write(1, receive, sizeof(receive));
65    // printf("%s\n", receive);
66  }
67
68    /* ソケットをクローズ */
69    close(sock);
70  }
```

A.3 (課題 2-2) simple_talk_server.c

```
 1  #include <sys/types.h>
 2  #include <sys/socket.h>
 3  #include <netinet/in.h>
 4  #include <netdb.h>
 5  #include <unistd.h>
 6  #include <string.h>
 7  #include <stdio.h>
 8  #include <stdlib.h>
 9  #define PORT 10130
10
11  int main(int argc, char **argv) {
12    int sock, csock; /* クライアントを受け付けたソケット*/
13    struct sockaddr_in svr;
14    struct sockaddr_in clt;
15    struct hostent *cp;
16    int clen;
```

```
17    char sbuf[1024], cbuf[1024];
18    int nbytes;
19    int reuse;
20    fd_set rfds; /* select() で用いるファイル記述子集合*/
21    struct timeval tv; /* select() が返ってくるまでの待ち時間を指定する変数*/
22
23    /* ソケットの生成 */
24    if ((sock = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP)) < 0) {
25      perror("socket");
26      exit(1);
27    }
28
29    /* ソケットアドレス再利用の指定 */
30    reuse = 1;
31    if(setsockopt(sock, SOL_SOCKET, SO_REUSEADDR, &reuse, sizeof(reuse)) < 0) {
32      perror("setsockopt");
33      exit(1);
34    }
35
36    /* client 受付用ソケットの情報設定 */
37    bzero(&svr, sizeof(svr));
38    svr.sin_family = AF_INET;
39    svr.sin_addr.s_addr = htonl(INADDR_ANY);
40
41    /* 受付側の IP アドレスは任意 */
42    svr.sin_port = htons(PORT); /* ポート番号 10130 を介して受け付ける */
43
44    /* ソケットにソケットアドレスを割り当てる */
45    if(bind(sock, (struct sockaddr *)&svr, sizeof(svr)) < 0) {
46      perror("bind");
47      exit(1);
48    }
49
50    /* 待ち受けクライアント数の設定 */
51    if(listen(sock, 5) < 0) { /* 待ち受け数に 5 を指定 */
52      perror("listen");
53      exit(1);
54    }
55    do {
56
57      /* クライアントの受付 */
58      clen = sizeof(clt);
59      if ((csock = accept(sock, (struct sockaddr *)&clt, &clen)) < 0) {
60        perror("accept");
61        exit(2);
62      }
63
64      /* クライアントのホスト情報の取得 */
65      cp = gethostbyaddr((char *)&clt.sin_addr, sizeof(struct in_addr), AF_INET);
66      printf("[%s]\n", cp->h_name);
67      printf("connected\n");
68
69      nbytes = -1;
70
71      do{
72        /* 入力を監視するファイル記述子の集合を変数 rfds にセットする*/
73        FD_ZERO(&rfds); /* rfds を空集合に初期化*/
74        FD_SET(0, &rfds); /* 標準入力*/
```

```
75        FD_SET(csock, &rfds); /* クライアントを受け付けたソケット*/
76        /* 監視する待ち時間を 1 秒に設定*/
77        tv.tv_sec = 1;
78        tv.tv_usec = 0;
79        memset(sbuf, '\0', sizeof(sbuf));
80        memset(cbuf, '\0', sizeof(cbuf));
81
82        /* 標準入力とソケットからの受信を同時に監視する */
83        if(select(csock+1, &rfds, NULL, NULL, &tv) > 0) {
84          if(FD_ISSET(0, &rfds)) { /* 標準入力から入力があったなら*/
85            if((nbytes = read(0, sbuf, sizeof(sbuf))) > 0){ /* 標準入力から読み込みクライ
アントに送信 */
86               write(csock, sbuf, nbytes);
87             }
88          }
89          if(FD_ISSET(csock, &rfds)) { /* ソケットから受信したなら*/
90          /* ソケットから読み込み端末に出力*/
91           if((nbytes = read(csock, cbuf, sizeof(cbuf))) > 0){
92              write(1, cbuf, nbytes);
93           }
94          }
95        }
96      } while (nbytes != 0);
97
98      /* read() が 0 を返すまで (=End-Of-File) 繰り返す */
99      close(csock);
100      printf("closed\n");
101    } while(1); /* 繰り返す*/
102  }
```

A.4 (課題 2-2) simple_talk_client.c

```
1  #include <sys/types.h>
2  #include <sys/socket.h>
3  #include <netinet/in.h>
4  #include <netdb.h>
5  #include <unistd.h>
6  #include <string.h>
7  #include <stdio.h>
8  #include <stdlib.h>
9  #define PORT 10130
10
11 int main(int argc, char **argv) {
12   int sock, nbytes;
13   char sbuf[1024], *name, cbuf[1024];
14   struct sockaddr_in svr;
15   struct hostent *hp;
16   fd_set rfds;
17   struct timeval tv;
18
19   if(argc != 2) {
20     fprintf(stderr, "Usage: %s hostname \n", argv[0]);
21     exit(1);
22   }else{
23     name = *(argv+1);
24   }
25
```

```
26    if((sock = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP)) < 0){
27      perror("socket");
28      exit(1);
29    }
30
31    bzero(&svr, sizeof(svr));
32    svr.sin_family = AF_INET;
33    svr.sin_port = htons(PORT);
34    if((hp = gethostbyname(name)) == NULL){
35      perror("gethostbyname");
36      exit(1);
37    }
38    bcopy(hp->h_addr, &svr.sin_addr, hp->h_length);
39
40    if((connect(sock, (struct sockaddr *)&svr, sizeof(svr))) < 0){
41        perror("connect");
42        exit(1);
43    }
44    else {
45      printf("connected\n" );
46    }
47
48    do{
49      FD_ZERO(&rfds);
50      FD_SET(0, &rfds);
51      FD_SET(sock, &rfds);
52      tv.tv_sec = 1;
53      tv.tv_usec = 0;
54      memset(sbuf, '\0', sizeof(sbuf));
55      memset(cbuf, '\0', sizeof(cbuf));
56
57      if(select(sock+1, &rfds, NULL, NULL, &tv) > 0){
58        if(FD_ISSET(sock, &rfds)){
59          if((nbytes = read(sock, cbuf, sizeof(cbuf))) > 0){
60            write(1, cbuf, nbytes);
61          }
62        }
63        if(FD_ISSET(0, &rfds)){
64          if((nbytes = read(0, sbuf, sizeof(sbuf))) > 0){
65            write(sock, sbuf, nbytes);
66          }
67        }
68      }
69    } while(nbytes != 0);
70    close(sock);
71    printf("closed\n");
72  }
```