情報科学演習C 第2レポート課題

担当教員：小島 英春、内山 彰 教員
提出者：椙村 渉
学籍番号：09B16036
メールアドレス：u739638g@ecs.osaka-u.ac.jp

提出年月日：2018 年 5 月 27 日

1 課題 1

1.1 目的

簡単な通信プログラムを作成し、UNIX と C 言語によるネットワークプログラミングの基礎を学習する。初めに、単純なパケットの送受信のみを行うプログラムを用いてネットワークプログラミングで、一般に必要とされるシステムコールなどを学習する。

1.2 課題内容

例示された UDP パケット送信プログラムと UDP パケット受信プログラムを実行し、実際に入力を行うことで動作を確かめる。その後コードリーディングを行うことでソケットプログラミングの基本を身につける。

1.3 実行結果

以下に udp_send.c と udp_receive.c をそれぞれ実際に実行し、その実行結果を記載する。

1.3.1 udp_send.c

udp_send.c の実行結果

```
w-sugimr@exp024:~/ensyuC/kadai2$ ./udp_send.out exp025 kadaisyuuryou
w-sugimr@exp024:~/ensyuC/kadai2$ ./udp_send.out exp025 kadais
w-sugimr@exp024:~/ensyuC/kadai2$ ./udp_send.out exp025 syuuryou
```

1.3.2 udp_receive.c

udp_receive.c の実行結果

```
w-sugimr@exp025:~/ensyuC/kadai2$ ./udp_receive.out
kadaisyuuryou from [192.168.16.46:40583]
kadais from [192.168.16.46:26536]
syuuryou from [192.168.16.46:36575]
```

上記の実行結果を確認すると、udp_send から送ったメッセージが送り元の情報と同時に udp_receive 側で標準出力されていることが確認できる。

1.4 考察

ここで考察として、与えれた 2 つの UDP プログラムがどのように実装されたかを以下にまとめる。送受信それぞれのプログラムは以下の流れで実装されている。

UDP パケット受信プログラムの流れ

1. socket 関数で通信の出入口として用いるソケットを生成する。
2. bind 関数でソケットとポートを対応付ける。
3. recvfrom 関数でデータを受信する。

UDP パケット送信プログラムの流れ

1. socket 関数で通信の出入口として用いるソケットを生成する。
2. gethostbyname 関数で宛先の IP アドレスを取得する。
3. sendto 関数でパケットを送出し、データを送信する。

上記流れに沿ってプログラムを作成することで UDP プログラムを実装することができる。

2 課題２－１

2.1 目的

サーバープログラムとクライアントプログラムの違いを学び、TCP サーバープログラミングの基本を身につける。

2.2 課題内容

与えられたサーバープログラム—echoserver.c と通信を行えるクライアントプログラム—echoclient.c を作成する。以下に本課題で作成するプログラムの詳細な仕様を記載する。また、telnet コマンドを用いて echoserver が正しく動作していることを確認する。

- サーバープログラムは host1 から次のようにして実行する。
  % .echoserver
- クライアントプログラムは host2 から次のようにして実行する。
  % .echoclient host1
- クライアントプログラム—echoclient で標準入力から読み込んだ文字列をサーバープログラム—echoserver に送信する。
- サーバープログラム—echoserver は受信した文字列に変更を加えることなく、そのままの状態でクライアントプログラム—echoclient に送り返す。

- クライアントプログラム—echoclient はサーバープログラム—echoserver から送り返された文字列を標準出力で出力する。
- この処理を標準入力から EOF を受け取る（Ctrl + D を入力）まで繰り返す。

2.3 作成プログラムの説明

以下に作成したクライアントプログラム—echoclient.c のプログラムの説明を行う。なおソースコード全体は付録として本レポート最後に記載する。

2.4 クライアントプログラム—echoclient.c

12-15 行目

```
12   int sock;
13   struct sockaddr_in host;
14   struct hostent *hp;
15   int nbytes;
16   char sbuf[1024];
17   char rbuf[1024];
```

ここでは変数、配列、構造体の宣言を行う。以下の表にそれぞれをプログラム上でどのように用いているかを示す。

型	変数名	使用用途
int 型	sock	送信用ソケットの生成に用いる
struct 型 sockaddr_in	host	クライアントのソケットの情報を保存する
struct 型 hostent	*hp	サーバーのホストの情報を保存する
int 型	nbytes	read 関数の返り値を保存する
char 型	sbuf	送信するメッセージを保存する
char 型	rbuf	受信するメッセージを保存する

表 1: echoclient.c の変数対応表

18- 21 行目

```
18   if(argc != 2){
19     fprintf(stderr,"Usage: %s hostname message\n",argv[0]);
20     exit(1);
21   }
```

ここではプログラムの実行の際に与えられた引数（ホスト名）が 1 つであるかの判別処理を行う。引数の数が 1 つ以外であった場合は標準出力でエラーであることを知らせ、プログラムを終了する。

23-27 行目

```
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   }
```

ここではソケットの生成を行う。socket 関数の第 1 引数には INET ドメインの AF_INET を与え、第 2 引数にはソケットの型を与える。今回作成するプログラムは TCP であるため SOCK_STREAM（ストリーム型）を与えるが、UDP の場合は SOCK_DGRAM（データグラム型）を与える。第 3 引数には用いるプロトコル（通信手段）の種類を与える。第 2 引数と同様に今回は TCP であるため IPPROTO_TCP を与えるが、UDP の場合は特に指定せず 0 を与える。そして socket 関数の返り値を int 型変数 sock に代入し、その値が 0 より小さい場合はエラーと判断し、プログラムを終了する。

29-37 行目

```
29   /*ソケットの情報設定*/
30   bzero(&host,sizeof(host));
31   host.sin_family = AF_INET;
32   host.sin_port = htons(10120);
33   if((hp = gethostbyname(argv[1])) == NULL){
34     fprintf(stderr,"unknown host %s\n",argv[1]);
35     exit(1);
36   }
37   bcopy(hp->h_addr, &host.sin_addr, hp->h_length);
```

ここではソケットの情報を構造体 host に保存する。まず 30 行目で構造体 host の中身を初期化し、31-32 行目でそれぞれ host のメンバ sin_family と sin_port に AF_INET とポート番号 10120 を保存する。32-3535 行目で gethostbyname 関数を用いてプログラム実行時に引数として与えられたホストのホスト名を取得し、変数 hp に保存する。37 行目で bcopy 関数を用いてそのホスト名を host のメンバ sin_addr に保存する。

39-44 行目

```
39   /*ソケットをサーバーに接続*/
40   if(connect(sock, (struct sockaddr *)&host, sizeof(host)) > 0){
41     perror("connect");
42     close(sock);
43     exit(1);
44   }
```

ここではソケットをサーバープログラムで bind されたホストに接続を行う。connect 関数を用いて、第 1 引数として sock を与え、第 2 引数として構造体 host 与えることで 29-37 行目で保存した

ホスト先を指定する。また返り値が 0 より小さい場合はエラーと判断し、プログラムを終了する。

45-59 行目

```
46   do{
47     char rbuf[1024] = {0};
48     char sbuf[1024] = {0};
49     scanf("%s", sbuf);
50     /*サーバーにメッセージの送信*/
51     write(sock, sbuf, strlen(sbuf));
52     /*サーバーからのメッセージの受信*/
53     if((nbytes = read(sock, rbuf, sizeof(rbuf))) < 0){
54       perror("read");
55     }
56     else{
57       printf("%s\n",rbuf);
58     }
59   }
60   while(sbuf != EOF);
```

ここでは接続後の通信でメッセージの送受信処理を行う。47-48 行目で送受信するメッセージを保存するそれぞれの配列を初期化する。49 行目で標準入力から送信するメッセージを入力する。51 行目で write 関数に sock と送信するメッセージを保存する sbuf を引数として与え、サーバープログラムに送信する。53 行目で read 関数に sock と受信するメッセージを保存する rbuf を引数として与え、サーバープログラムからメッセージを受信する。この時の返り値が 0 より小さい場合はエラーと判断し、エラーを出力する。またこの時 0 以上の場合、標準出力で受信したメッセージを出力する。

2.4.1 実行結果

まずは telnet コマンドを用いて正常に echoserver が動作しているかを確認する。その実行結果を如何に記載する。

telnet コマンド実行結果

```
w-sugimr@exp024:~/ensyuC/kadai2$ telnet exp025 10120
Trying 192.168.16.52...
Connected to exp025.exp.ics.es.osaka-u.ac.jp.
Escape character is '^]'.
```

上記の実行結果より、echoserver がポート番号 10120 を使用し、正常に動作していることが確認できる。

次に実際にそれぞれのホストにリモートログインを行い、サーバープログラムとクライアントプログラムを実行してみた。その実行結果を以下に記載する。

echoserver.c の実行結果

```
w-sugimr@exp025:~/ensyuC/kadai2$ ./echoserver.exe exp024
[exp024.exp.ics.es.osaka-u.ac.jp]
closed
[exp024.exp.ics.es.osaka-u.ac.jp]
```

上記の実行結果を確認すると、ホスト名 exp003 のサーバープログラムにホスト名 exp004 のクライアントプログラムから接続ができていることが確認でき、一度接続を切断すると closed と標準出力し、次のクライアントプログラムを受け付けていることも確認できる。

echoclient.c の実行結果

```
w-sugimr@exp024:~/ensyuC/kadai2$ ./echoclient.exe exp025
kadai
kadai
syuuryou
syuuryou
```

上記の実行結果を確認すると、サーバープログラムに接続後、標準入力からメッセージを入力すると、そのメッセージが折り返し送信され、受信したメッセージを標準出力で出力がされていることが確認できる。

2.5 考察

ここまでで本課題の目的である TCP サーバープログラミングの実装を行った。ここで改めて学習した内容を大まかにまとめ、その内容をどのように実装したかを考察としてまとめる。

まずサーバープログラムとクライアントプログラムの大きな違いをまとめ、以下に記載する。

サーバープログラム

サーバープログラムはクライアントプログラムからの接続要求を待つ。

クライアントプログラム

クライアントプログラムは接続待ちをしているサーバープログラムに対して接続要求を出し、接続が成功すると通信を開始する。

それぞれ一度接続が完了するとサーバープログラムとクライアントプログラム間で通信方法の大きな相違は存在しない。そのため主な違いとしては、接続を開始する前の段階でプログラム内容がそれぞれ異なることが理解できる。

そのプログラム内容の違いをまとめ、以下に記載する。

サーバープログラムの流れ

1. socket 関数で通信の出入口として用いるソケットを生成する。
2. bind 関数でソケットとポートを対応付ける。

3. listen 関数で最大接続待ち数を指定する。
4. accept 関数でクライアントからの接続要求を待つ。
   到着したら新たにソケットを生成する。
5. 接続後、通信を行う。
6. close 関数で新たに生成したソケットを閉じる。

クライアントプログラムの流れ

1. socket 関数で通信の出入口として用いるソケットを生成する。
2. connect 関数でサーバープログラムで bind された host と接続する。
3. 接続後、通信を行う。
4. close 関数で通信を終了する。

2.3 のプログラムの説明を確認すると実際に上記プログラムの流れに沿って実装が完了しており、課題内容と目的を満たしていることが確認できる。

3 課題２－２

3.1 目的

他のユーザーと 1 対 1 の通信を行う talk というプログラムの必要最小限の機能のみを実現することで、全二重通信の実現方法やソケットプログラミングの応用を身につける。

3.2 課題内容

他のユーザーと 1 対 1 の通信を行うプログラムを作成する。サーバープログラム―simple-talk-server.c とクライアントプログラム―simple-talk-client.c に分け、それぞれから並行して文字列を入力して相手の端末にコピーするという仕様の実装を行う。また以下に本課題で作成するプログラムの詳細な仕様を記載する。

- サーバープログラムは host1 から次のようにして実行する。
  % .simple-talk-server
- クライアントプログラムは host2 から次のようにして実行する。
  % .simple-talk-client host1
- サーバープログラム－ simple-talk-server とクライアントプログラム－ simple-talk-client はそれぞれ host1 と host2 の端末上に”connected”とメッセージを出力し、通信が確立したことを知らせる。
- 接続の確立後、一方のユーザーの端末から入力された文字列を他方のユーザーの端末上にコピーされる。会話を行う二人がどのような順番で発言を行っても、即時にそのメッセージが相手に表示され、全二重通信で実装。

- 受信を行ったメッセージは標準出力で">> （メッセージ）"と表示される。
- どちらか一方の接続が切れた時、それぞれプログラムを終了させる（サーバープログラムの場合は他のクライアントからの接続待機に移行）。

3.3 作成プログラムの説明

ここでの説明はこれまでの課題 1 と課題 2 − 1 で説明した内容は自明として説明を行う。

3.3.1 サーバープログラムーsimple-talk-server.c

以下に作成したサーバープログラムーsimple-talk-server.c のプログラムの説明を行う。なおソースコード全体は付録として本レポート最後に記載する。

12-19 行目

```
12   int sock,csock;
13   struct sockaddr_in svr, clt;
14   struct hostent *cp;
15   int clen;
16   char rbuf[1024], sbuf[1024];
17   int reuse;
18   fd_set rfds; /* select() で用いるファイル記述子集合 */
19   struct timeval tv; /* select() が返ってくるまでの待ち時間を指定する変数 */
```

ここでは変数、配列、構造体の宣言を行う。以下の表にそれぞれをプログラム上でどのように用いているかを示す。

型	変数名	使用用途
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
struct 型 sockaddr_in	svr	クライアント受信用のソケットの情報を保存する
struct 型 sockaddr_in	clt	クライアントのソケットの情報を保存する
struct 型 hostent	*cp	クライアントホストの情報を保存する
char 型配列	rbuf[1024]	受信したメッセージを保存する
char 型配列	sbuf[1024]	送信するメッセージを保存する
int 型	reuse	ソケットの再利用の際に用いる
fd_set 型	rfds	ファイルディスクリプタの集合を保存する
struct 型 timeval	tv	select 関数を監視する待ち時間を保存する構造体

表 2: simple-talk-server.c の変数対応表

65-71 行目

```
65        /* 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 */
66        FD_ZERO(&rfds); /* rfds を空集合に初期化 */
67        FD_SET(0,&rfds); /* 標準入力 */
68        FD_SET(csock,&rfds); /* クライアントを受け付けたソケット */
69        /* 監視する待ち時間を 1 秒に設定 */
70        tv.tv_sec = 1;
71        tv.tv_usec = 0;
```

ここでは入力を監視するファイルディスクリプタの集合を変数 rfds に保存する。まず、66 行目で FD_ZERO マクロを用いて rfds に保存されている集合を空にする。次に 67 行目で FD_SET マクロの第 1 引数に 0 を与えることで標準入力を、68 行目で受信用ソケットである csock を与えることでクライアントプログラムからの受信を代入する。また 70—71 行目で入力を監視する待ち時間を 1 秒に設定を行う。

72-73 行目

```
72        /* 標準入力とソケットからの受信を同時に監視する */
73        if(select(csock+1, &rfds, NULL,NULL, &tv) > 0) {
```

ここでは 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する。select 関数には第 1 引数として監視するファイルディスクリプタの最大値に 1 を加えた整数を与える。本プログラムにおいてはクライアントからの受信するソケット csock に 1 を加えた値を与える。また第 2-4 引数にはそれぞれ、読み込み、書き込み、例外発生を監視するファイルディスクリプタの集合を指すポインタ与えるが、本プログラムにおいては読み込み処理のみを監視するため、第 3-4 引数には NULL ポインタを与える。第 5 引数には監視する待ち時間を保存した構造体へのポインタを与える。そのため本プログラムにおいては 70-71 行目で設定した構造体 tv のポインタを与える。そして返り値として監視しているファイルディスクリプタの状態が変化した総数が与えられ、エラーの場合は-1 が返る。if 条件文によって返り値が正の場合、エラーが発生していないと判断し、それぞれに対応する処理を実行する。

74-79 行目

```
74          if(FD_ISSET(0, &rfds)) { /* 標準入力から入力があったなら */
75            /* 標準入力から読み込みクライアントに送信 */
76            char sbuf[1024] = {0};
77            read(0, sbuf, sizeof(sbuf));
78            write(csock, sbuf, strlen(sbuf));
79          }
```

ここでは標準入力から入力があった場合の処理を行う。74 行目で if 条件文によって FD_ISSET マクロの第 1 引数に 0 を与えた時の返り値が真であった場合、標準入力から入力があったと判断し、75-77 行目の処理を実行する。76 行目で char 型配列 sbuf の中身を初期化し、77 行目で read 関数を用いて標準入力から入力された文字列を sbuf に代入する。最後に 78 行目で受信先（送信先）ソケットの csock に write 関数を用いてクライアントプログラムに送信を行う。

80-90 行目

```
80          if(FD_ISSET(csock, &rfds)) { /* ソケットから受信したなら */
81            /* ソケットから読み込み端末に出力 */
82            char rbuf[1024] = {0};
83            if(recv(csock, rbuf, sizeof(rbuf), 2) == 0) break;
84            if(read(csock, rbuf, sizeof(rbuf)) < 0){
85              perror("read");
86            }
87            else{
88              printf(">> %s", rbuf);
89            }
90          }
```

ここではクライアントプログラムからメッセージを受信した場合の処理を行う。80 行目で if 条件文によって FD_ISSET マクロの第 1 引数に csock を与えた時の返り値が真であった場合、クライアントプログラムからメッセージを受信したと判断し、81-89 行目の処理を実行する。82 行目で char 型配列 rbuf の中身を初期化し、84 行目で read 関数を用いてクライアントプログラムから受信した文字列を rbuf に代入する。この時 if 条件文によって返り値が 0 以上であった場合は 87-89 行目を実行し、88 行目で受信したメッセージの標準出力を行う。返り値が 0 以上ではなかった場合はエラーが発生したと判断し、エラーを出力する。また 83 行目で recv 関数を用いて、クライアントプログラムとの接続が切断されていないかを判別する。もしここで返り値が 0 であった場合はクライアントプログラムとの接続が切断されたと判断し、64-92 行目の do-while ループ処理を抜け、93-95 行目の処理を実行し、新たに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受ける処理に戻る（クライアントプログラム接続前に）。

93-95 行目

```
93      /* read() が 0 を返すまで (=End-Of-File) 繰り返す */
94      close(csock);
95      printf("closed\n");
```

ここでは上記でも述べたように、クライアントプログラムとの接続が切断された場合の処理を行う。94 行目で受信していたクライアントプログラムのソケットを閉じ、閉じたことを知らせるために標準出力で”closed”を出力する。

これらの処理を 53-96 行目で do-while ループ処理を繰り返すことで、本課題の仕様を実装した。

3.3.2 クライアントプログラムーsimple-talk-client.c

以下に作成したクライアントプログラムーsimple-talk-client.c のプログラムの説明を行う。しかし、サーバープログラムーsimple-talk-server.c とコードが殆ど重複するため、相違点のみの説明を行う。なおソースコード全体は付録として本レポート最後に記載する。

13-19 行目

```
13   int sock;
14   struct sockaddr_in host;
15   struct hostent *hp;
16   char rbuf[1024];
17   char sbuf[1024];
18   fd_set rfds; /* select() で用いるファイル記述子集合 */
19   struct timeval tv; /* select() が返ってくるまでの待ち時間を指定する変数 */
```

ここでは変数、配列、構造体の宣言を行う。以下の表にそれぞれをプログラム上でどのように用いているかを示す。

型	変数名	使用用途
int 型	sock	送受信用ソケットの生成に用いる
struct 型 sockaddr_in	host	ソケットの情報を保存する
struct 型 hostent	*hp	サーバーのホストの情報を保存する
char 型配列	rbuf[1024]	受信したメッセージを保存する
char 型配列	sbuf[1024]	送信するメッセージを保存する
fd_set 型	rfds	ファイルディスクリプタの集合を保存する
struct 型 timeval	tv	select 関数を監視する待ち時間を保存する構造体

表 3: simple-talk-client.c の変数対応表

50-79 行目

```
50   do{
51     /* 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 */
52     FD_ZERO(&rfds); /* rfds を空集合に初期化 */
53     FD_SET(0, &rfds); /* 標準入力 */
54     FD_SET(sock, &rfds); /* クライアントを受け付けたソケット */
55     /* 監視する待ち時間を 1 秒に設定 */
56     tv.tv_sec = 1;
57     tv.tv_usec = 0;
58     /* 標準入力とソケットからの受信を同時に監視する */
59     if(select(sock+1, &rfds, NULL, NULL, &tv) > 0){
60       if(FD_ISSET(0, &rfds)) { /* 標準入力から入力があったなら */
61         /* 標準入力から読み込みクライアントに送信 */
62         char sbuf[1024] = {0};
63         read(0, sbuf, sizeof(sbuf));
64         write(sock, sbuf, strlen(sbuf));
65       }
```

```
66        if(FD_ISSET(sock, &rfds)) { /* ソケットから受信したなら */
67          /* ソケットから読み込み端末に出力 */
68          char rbuf[1024] = {0};
69          if(recv(sock, rbuf, sizeof(rbuf), 2) == 0) exit(1);
70          if(read(sock, rbuf, sizeof(rbuf)) < 0) {
71            perror("read");
72          }
73          else{
74            printf(">> %s", rbuf);
75          }
76        }
77      }
78    }
79    while(1);
```

54、59、64、66、69、70 行目でそれぞれ引数として与えるソケットはサーバープログラムと異なり、sock を与える。これはクライアントプログラムが接続の際にソケットをサーバーに接続させる処理のみを行うため、サーバープログラムとことなら受信用のソケットを用意する必要がないからである。そのためクライアントプログラムで用いられているソケットは sock の 1 種類のみである。また 68 行目では recv 関数を用いて、サーバープログラムとの接続が切断されていないかを判別する。もしここで返り値が 0 であった場合はサーバープログラムとの接続が切断されたと判断し、プログラムを終了させる。これはサーバープログラムと異なり、他のクライアントプログラムの受信を待ち受ける必要がないため、サーバープログラムとの接続が切断した段階でこれ以上処理を行う内容が存在しないためである。

3.4 実行結果

実際にそれぞれのホストにリモートログインを行い、サーバープログラムとクライアントプログラムを実行してみた。その実行結果を以下に記載する。

simple-talk-server.c の実行結果

```
w-sugimr@exp003:~/ensyuC/kadai2$ ./simple-talk-server.exe
[exp004.exp.ics.es.osaka-u.ac.jp]
connected
setuzoku
kannryou
>> dousakakuninn
>> kokode
>> setuzokuwokiruto
closed
[exp004.exp.ics.es.osaka-u.ac.jp]
connected
```

```
>> saisetuzoku
konndoha
kotirakara
setuzokuwo
kiruto
^C
w-sugimr@exp003:~/ensyuC/kadai2$
```

simple-talk-client.c の実行結果

```
w-sugimr@exp004:~/ensyuC/kadai2$ ./simple-talk-client.exe exp003
connected
>> setuzoku
>> kannryou
dousakakuninn
kokode
setuzokuwokiruto
^C
w-sugimr@exp004:~/ensyuC/kadai2$ ./simple-talk-client.exe exp003
connected
saisetuzoku
>> konndoha
>> kotirakara
>> setuzokuwo
>> kiruto
w-sugimr@exp004:~/ensyuC/kadai2$
```

上記の実行結果を確認すると、ホスト名 exp003 のサーバープログラムにホスト名 exp004 のクライアントプログラムから接続ができていることが確認でき、またそれぞれのメッセージが送受信されていることが確認できる。まず初めに接続が完了したと同時に connected と標準出力されているため、接続を知らせる仕様を満たしていることが確認できる。また”>> （メッセージ）”と連続で出力されていることから、全二重通信での実装も満たせていることが確認できる。最後にクライアントプログラム側から接続を切断した場合、サーバープログラム側では再びクライアントプログラムの受信状態になり、再度接続を行うと同様に通信が可能になっていることが確認できる。逆にサーバープログラム側から接続を切断した場合、クライアントプログラムは強制的に終了していることも確認できる。

以上より、本課題における仕様はすべて満たしていると判断できる。

3.5 考察

ここまでで本課題の目的である全二重通信の実装は完了していることが確認できる。しかし、本課題で作成したプログラムには様々な制限がある。例えば、どちらか一方のプログラムでメッセージを標準入力から入力中の際に、他方のプログラムからメッセージを受信してしまった場合、それ

までに入力していたメッセージと受信したメッセージを同時に表示してしまう。
以下に例を記載する。

simple-talk-server.c 側

```
w-sugimr@exp003:~/ensyuC/kadai2$ ./simple-talk-server.exe
[exp004.exp.ics.es.osaka-u.ac.jp]
connected
>> tatoeba
okuruto
konoyouni
hyouzisareru
```

simple-talk-client.c 側

```
w-sugimr@exp004:~/ensyuC/kadai2$ ./simple-talk-client.exe exp003
connected
tatoeba
konotokini>> okuruto
>> konoyouni
>> hyouzisareru
```

上記の例を確認すると、クライアントプログラムで「konotokini」を入力途中にサーバープログラムから「okuruto」というメッセージを受信した結果、「konotokini>> okuruto」と表示されていることが確認できる。

今回作成したプログラムはあくまでも簡易 talk プログラムであるため、最低限の仕様を満たしている点で問題はないと考えられるが、実用レベルのプログラムとは程遠いと言える。何度か改善を試みたが、そもそもの端末の仕様上不可能であるのか、改善案は見つからなかった。この改善策を見つけることを課題とし、今後の学習に臨みたい。

4 感想

ネットワークの知識はほとんどゼロの状態から始めたため、課題を満たすべく非常に苦労した。だが、一度ソケットに関する知識の理解が深まると、思っていたよりもあっさりプログラムを完成させることができた。特に課題２－１を理解した後に課題２－２に取り掛かったが、select 関数の知識を身につけるだけでスラスラと理解できた。更に select 関数に関して調べていると、今回作成した最低限の仕様以外にも様々な仕様を満たせるのではないかと思い、学習していてとても楽しかった。次回以降の課題でも学習する楽しさを理解し、更なる知識の吸収を行いたい。

5 謝辞

課題内容を記載した資料は非常にわかりやすく、課題を進めるうえで非常に役に立ちました。残念ながら発展課題まで手を付ける余裕はありませんでしたが、課題提出後にでも個人的学習として

吸収できるものはしたいと考えています。お世話になりました。

6　参考文献

A　付録：作成プログラム

A.1　課題２－１－echoclient.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
12   int sock;
13   struct sockaddr_in host;
14   struct hostent *hp;
15   int nbytes;
16   char sbuf[1024];
17   char rbuf[1024];
18   if(argc != 2){
19     fprintf(stderr,"Usage: %s hostname message\n",argv[0]);
20     exit(1);
21   }
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   /*ソケットの情報設定*/
30   bzero(&host,sizeof(host));
31   host.sin_family = AF_INET;
32   host.sin_port = htons(10120);
33   if((hp = gethostbyname(argv[1])) == NULL){
34     fprintf(stderr,"unknown host %s\n",argv[1]);
35     exit(1);
36   }
37   bcopy(hp->h_addr, &host.sin_addr, hp->h_length);
38
39   /*ソケットをサーバーに接続*/
```

```
40   if(connect(sock, (struct sockaddr *)&host, sizeof(host)) > 0){
41     perror("connect");
42     close(sock);
43     exit(1);
44   }
45
46   do{
47     char rbuf[1024] = {0};
48     char sbuf[1024] = {0};
49     scanf("%s", sbuf);
50     /*サーバーにメッセージの送信*/
51     write(sock, sbuf, strlen(sbuf));
52     /*クライアントからのメッセージの受信*/
53     if((nbytes = read(sock, rbuf, sizeof(rbuf))) < 0){
54       perror("read");
55     }
56     else{
57       printf("%s\n",rbuf);
58     }
59   }
60   while(sbuf != EOF);
61
62   close(sock);
63   exit(1);
64 }
```

A.2 課題２－２－simple-talk-server.c

```
 1 #include <sys/types.h>
 2 #include <sys/time.h>
 3 #include <sys/socket.h>
 4 #include <netinet/in.h>
 5 #include <netdb.h>
 6 #include <unistd.h>
 7 #include <string.h>
 8 #include <stdio.h>
 9 #include <stdlib.h>
10
11 int main(int argc,char **argv) {
12   int sock,csock;
13   struct sockaddr_in svr, clt;
14   struct hostent *cp;
15   int clen;
```

```
16  char rbuf[1024], sbuf[1024];
17  int reuse;
18  fd_set rfds; /* select() で用いるファイル記述子集合 */
19  struct timeval tv; /* select() が返ってくるまでの待ち時間を指定する変数 */
20
21  /* ソケットの生成 */
22  if ((sock = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP)) < 0) {
23    perror("socket");
24    exit(1);
25  }
26
27  /* ソケットアドレス再利用の指定 */
28  reuse=1;
29  if(setsockopt(sock, SOL_SOCKET, SO_REUSEADDR, &reuse, sizeof(reuse)) < 0) {
30    perror("setsockopt");
31    exit(1);
32  }
33
34  /* client 受付用ソケットの情報設定 */
35  bzero(&svr, sizeof(svr));
36  svr.sin_family = AF_INET;
37  svr.sin_addr.s_addr = htonl(INADDR_ANY);
38  /* 受付側の IP アドレスは任意 */
39  svr.sin_port = htons(10130); /* ポート番号 10130 を介して受け付ける */
40
41  /* ソケットにソケットアドレスを割り当てる */
42  if(bind(sock, (struct sockaddr *)&svr, sizeof(svr))<0) {
43    perror("bind");
44    exit(1);
45  }
46
47  /* 待ち受けクライアント数の設定 */
48  if (listen(sock, 5)<0) { /* 待ち受け数に 5 を指定 */
49    perror("listen");
50    exit(1);
51  }
52
53  do {
54    /* クライアントの受付 */
55    clen = sizeof(clt);
56    if ((csock = accept(sock, (struct sockaddr *)&clt, &clen) ) < 0) {
57      perror("accept");
58      exit(2);
```

```
59      }
60      /* クライアントのホスト情報の取得 */
61      cp = gethostbyaddr((char *)&clt.sin_addr, sizeof(struct in_addr), AF_INET);
62      printf("[%s]\n", cp->h_name);
63      printf("connected\n");
64      do {
65        /* 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 */
66        FD_ZERO(&rfds); /* rfds を空集合に初期化 */
67        FD_SET(0,&rfds); /* 標準入力 */
68        FD_SET(csock,&rfds); /* クライアントを受け付けたソケット */
69        /* 監視する待ち時間を 1 秒に設定 */
70        tv.tv_sec = 1;
71        tv.tv_usec = 0;
72        /* 標準入力とソケットからの受信を同時に監視する */
73        if(select(csock+1, &rfds, NULL,NULL, &tv) > 0) {
74          if(FD_ISSET(0, &rfds)) { /* 標準入力から入力があったなら */
75            /* 標準入力から読み込みクライアントに送信 */
76            char sbuf[1024] = {0};
77            read(0, sbuf, sizeof(sbuf));
78            write(csock, sbuf, strlen(sbuf));
79          }
80          if(FD_ISSET(csock, &rfds)) { /* ソケットから受信したなら */
81            /* ソケットから読み込み端末に出力 */
82            char rbuf[1024] = {0};
83            if(recv(csock, rbuf, sizeof(rbuf), 2) == 0) break;
84            if(read(csock, rbuf, sizeof(rbuf)) < 0){
85              perror("read");
86            }
87            else{
88              printf(">> %s", rbuf);
89            }
90          }
91        }
92      } while (1);
93      /* read() が 0 を返すまで (=End-Of-File) 繰り返す */
94      close(csock);
95      printf("closed\n");
96    } while(1); /* 次の接続要求を繰り返し受け付ける */
97  }
```

A.3 課題２－２－simple-talk-client.c

```
1 #include <sys/types.h>
```

```
 2 #include <sys/time.h>
 3 #include <sys/socket.h>
 4 #include <netinet/in.h>
 5 #include <netdb.h>
 6 #include <unistd.h>
 7 #include <string.h>
 8 #include <stdio.h>
 9 #include <stdlib.h>
10
11 int main(int argc, char **argv){
12
13   int sock;
14   struct sockaddr_in host;
15   struct hostent *hp;
16   char rbuf[1024];
17   char sbuf[1024];
18   fd_set rfds; /* select() で用いるファイル記述子集合 */
19   struct timeval tv; /* select() が返ってくるまでの待ち時間を指定する変数 */
20
21   if(argc != 2){
22     fprintf(stderr,"Usage: %s hostname message\n",argv[0]);
23     exit(1);
24   }
25
26   /*ソケットの生成*/
27   if((sock = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP)) < 0){
28     perror("socket");
29     exit(1);
30   }
31
32   /*ソケットの情報設定*/
33   bzero(&host, sizeof(host));
34   host.sin_family = AF_INET;
35   host.sin_port = htons(10130);
36   if((hp = gethostbyname(argv[1])) == NULL){
37     fprintf(stderr,"unknown host %s\n",argv[1]);
38     exit(1);
39   }
40   bcopy(hp->h_addr, &host.sin_addr, hp->h_length);
41
42   /*ソケットをサーバーに接続*/
43   if(connect(sock, (struct sockaddr *)&host, sizeof(host)) > 0){
44     perror("connect");
```

```
45      close(sock);
46      exit(1);
47    }
48    else printf("connected\n");
49
50    do{
51      /* 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 */
52      FD_ZERO(&rfds); /* rfds を空集合に初期化 */
53      FD_SET(0, &rfds); /* 標準入力 */
54      FD_SET(sock, &rfds); /* クライアントを受け付けたソケット */
55      /* 監視する待ち時間を 1 秒に設定 */
56      tv.tv_sec = 1;
57      tv.tv_usec = 0;
58      /* 標準入力とソケットからの受信を同時に監視する */
59      if(select(sock+1, &rfds, NULL, NULL, &tv) > 0){
60        if(FD_ISSET(0, &rfds)) { /* 標準入力から入力があったなら */
61          /* 標準入力から読み込みクライアントに送信 */
62          char sbuf[1024] = {0};
63          read(0, sbuf, sizeof(sbuf));
64          write(sock, sbuf, strlen(sbuf));
65        }
66        if(FD_ISSET(sock, &rfds)) { /* ソケットから受信したなら */
67          /* ソケットから読み込み端末に出力 */
68          char rbuf[1024] = {0};
69          if(recv(sock, rbuf, sizeof(rbuf), 2) == 0) exit(1);
70          if(read(sock, rbuf, sizeof(rbuf)) < 0) {
71            perror("read");
72          }
73          else{
74            printf(">> %s", rbuf);
75          }
76        }
77      }
78    }
79    while(1);
80
81    close(sock);
82  }
```