情報科学演習C 課題2資料

文責：内山 彰 *

2018年4月16日 (月)

1 目的

簡単な通信プログラムを作成することにより，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ぶ．

最初に，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一般に必要となるシステムコールなどを学習し，その後，高機能な通信プロトコルを利用するためのシステムコールを使ったプログラムを作成する．

2 ネットワークプログラミングの基礎 — ソケットについて

UNIXとC言語によるプロセス間通信では**ソケット**という仕組みが用いられる．ソケットはプロセスと通信路を結ぶ接続点であり，C言語ではsocket()システムコール[1]によって生成される．ソケットを介してどのような通信を行うかに応じてソケットを生成する際に主にUDPとTCPという2つのプロトコルが選択できる．

UDPはインターネットにおいて最も基本的な通信プロトコルであり，主にパケットを送信する機能と受信する機能のみが提供される．シンプルであるがゆえに，プログラム作成者が通信の細かい部分まで制御可能であり，特に通信速度が必要なプログラムなどで用いられる傾向にある．送出されたパケットが何らかの理由で相手に届かなかった場合[2] もパケットが失われたことは通知されない．確実に相手までパケットを届けたい場合には，届いたかどうかのチェックをするコードや，届けられなかった場合に再送を行うコードなどをプログラマが必要に応じてプログラムする必要がある．

一方，TCPはインターネットにおいて最もよく利用されているプロトコルであり，UDPよりも複雑な仕組みで，より確実な通信機能を提供する．TCPでは，送信されたデータは送信された順序で相手に到着し，通信の経路上でデータが失われたような場合にはOSが自動的にデータを再送し，確実なデータの到着を保証する[3]．そのため，プログラマは詳細を気にせずプログラムを行う

---

*本稿の作成に当たり，小島先生が書かれた2017年度演習C課題2を参考にした．

[1] C言語であらかじめ用意されている関数には**システムコール** (OSが提供する機能を直接呼び出す関数) と**ライブラリコール** (C言語で標準的に用意されたコンパイル済みの関数) があり，socket()などは前者に属する．前者はOSに密着した基本的な機能を提供するのに対して，後者は一般に前者を組み合わせたより高度な機能を提供する．本課題のようなソケットを用いたプログラミングでは主にシステムコールが用いられる．他の環境ではAPI(Application Program Interface)などとも呼ばれる．

[2] 通信路上の物理的なノイズや機器の故障，輻輳の発生など理由は多岐にわたる．

[3] ただし，修復しようのない故障などの場合には，もちろん通信が途絶する．その場合はプログラムに対して，通信が途絶した，という事実が知らされるため，プログラマはその事態に対応するためのコードを用意しておく必要がある．

ことができる．TCP 用に生成したソケットに対しては，ファイルの読み書きと同じ read(),write() システムコールを用いて入出力を行うことにより通信を実現する．

2.1 UDP 通信プログラム例

まず最初に，ごく単純な UDP パケット送受信プログラムを例示する．UDP パケットの送信は以下の手順で行う．

1. socket() で通信の出入口となるソケットを生成する．
2. gethostbyname() で宛先の IP アドレスを取得する．
3. sendto() で宛先にパケットを送出する．

UDP パケットを受信する場合は以下の手順となる．

1. socket() で通信の出入口となるソケットを生成する．
2. bind() で socket に待ち受けするネットワークインターフェイスを割り当てる．
3. recvfrom() で受信する．

具体的なソースコードを以下に示す．

2.1.1 UDP パケット送信プログラム — udp_send.c

```
#include <sys/types.h>     /* socket() を使うために必要 */
#include <sys/socket.h>    /*  同上    */
#include <netinet/in.h>    /* INET ドメインのソケットを使うために必要 */
#include <netdb.h>         /* gethostbyname() を使うために必要 */
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>

#define PORT 10000

int main(int argc,char **argv)
{
  int sock;
  struct sockaddr_in host;
  struct hostent *hp;
  if (argc != 3) {
    fprintf(stderr,"Usage: %s hostname message\n",argv[0]);
    exit(1);
```

```
  }

  /* ソケットの生成 */
  if ( ( sock = socket(AF_INET,SOCK_DGRAM,0) ) < 0) {
    perror("socket");
    exit(1);
  }

  /* host(ソケットの接続先) の情報設定 */
  bzero(&host,sizeof(host));
  host.sin_family=AF_INET;
  host.sin_port=htons(PORT);

  if ( ( hp = gethostbyname(argv[1]) ) == NULL ) {
    fprintf(stderr,"unknown host %s\n",argv[1]);
    exit(1);
  }
  bcopy(hp->h_addr,&host.sin_addr,hp->h_length);

  /* パケットの送信 */
  sendto(sock, argv[2], strlen(argv[2]), 0, (struct sockaddr*)&host,sizeof(host));

  close(sock);
  exit(0);
}
```

2.1.2 UDP パケット受信プログラム — udp_receive.c

```
#include <sys/types.h>     /* socket() を使うために必要 */
#include <sys/socket.h>    /*  同上   */
#include <netinet/in.h>    /* INET ドメインのソケットを使うために必要 */
#include <arpa/inet.h>     /*  同上   */
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>

#define PORT 10000

int main(void)
{
  int sock;
```

```
struct sockaddr_in svr;
struct sockaddr_in sender;
int sender_len;
struct hostent *hp;
char buf[1024];
char senderstr[1024];
int len;

/* ソケットの生成 */
if ( ( sock = socket(AF_INET,SOCK_DGRAM,0) ) < 0) {
  perror("socket");
  exit(1);
}

/* client 受付用ソケットの情報設定 */
bzero(&svr,sizeof(svr));
svr.sin_family=AF_INET;
svr.sin_addr.s_addr=htonl(INADDR_ANY);
/* ホストの IP アドレスは一般に複数あり各ネットワークインタフェース
   (通信の受け口となるハードウェアあるいは仮想ハードウェア
   (ループバックインタフェースなど)) に割り当てられている.
   クライアントがどこからアクセスするかによって
   サーバのどのインタフェースがそのどれから受信してもいいように
   ワイルドカードとして INADDR_ANY を指定する.   */
svr.sin_port=htons(PORT);

/* ソケットにソケットアドレスを割り当てる */
if(bind(sock,(struct sockaddr*)&svr,sizeof(svr))<0) {
  perror("bind");
  exit(1);
}

/* パケットの受信と表示 */
while(1){
  // 受け入れるパケットの送信元を指定 (0.0.0.0 で任意のアドレスからのパケットを受信)
  bzero(&sender,sizeof(sender));
  sender_len = sizeof(sender);
  if((len = recvfrom(sock, buf, sizeof(buf), 0,
                     (struct sockaddr*)&sender, &sender_len)) < 0){
    perror("recvfrom");
    exit(1);
```

```
    }

    inet_ntop(AF_INET, &sender.sin_addr, senderstr, sizeof(senderstr));
    write(1, buf, len);
    printf(" from [%s:%d]\n", senderstr, sender.sin_port);
  }
}
```

2.1.3 プログラムの説明

udp_receive は，実行すると UDP パケットがポート 10000 に到着するまで待機する．一般にホストでは複数のプロセスが稼働しており，IP アドレスだけではそのホストのどのプロセスと通信してよいかわからない．通信相手のプロセスを識別するため**ポート番号**というものが用いられる[4]．パケットが到着し次第，そのパケットの送信元と，パケットの内容を表示する．表示した後，再び待機に戻る．

udp_send は，2 つの引数を取るプログラムである．1 つ目の引数が送信先のホスト名 (IP アドレスでも良い)，2 つ目の引数が送信するメッセージである．例えば，

```
% udp_send exp101 hello
```

のように実行すると，ホスト exp101 のポート 10000 に対して，“hello” という内容の UDP パケットが送出される．exp101 で udp_receive が動作していた場合には，udp_receive が hello というメッセージを表示し，UDP パケットの到着が分かる．

まず，以上のプログラムを実際に入力して動作を確かめることで，ソケットプログラミングの基本を身につけよ．また，余力のある者は P.13 の発展課題 1 にも挑戦せよ．以下にプログラム中で用いられているシステムコールに関して説明する．

2.1.4 ソケットの生成 (共通)

socket() システムコールを用いてソケットを生成する[5]．socket() の第 1 引数にはヘッダファイル<sys/socket.h>で定義されたドメインの種類（INET ドメインでは AF_INET）を指定する．第 2 引数には同じく<sys/socket.h>で定義されたソケットの型を指定する．ソケットの型には**ストリーム型**，**データグラム型**などがあり，UDP はデータグラム型の通信であるため，ここではデータグラム型（SOCK_DGRAM）を指定する．第 3 引数には用いる**プロトコル** (通信手順) の種類を指定する．UDP の場合は特に指定する必要は無いため，0 としておく．詳細はオンラインマニュアル (man 2 socket) を参照のこと．

[4]標準的なサービスに対してはあらかじめポート番号が割り当てられており，その対応は UNIX では通常/etc/services に記述してある．例えば，http サービスにはポート番号 80 が割り当てられている．

[5]ソケットには大きく分けて **UNIX ドメイン**のソケットと **INET ドメイン**のソケットの 2 種類がある．UNIX ドメインのソケットは同じホストで動いているプロセス同士で通信するためのソケットであり，INET ドメインのソケットはネットワークを介して離れたホストで稼働しているプロセスと通信するためのソケットである．ここでは INET ドメインのソケットを用いる．

2.1.5 gethostbyname() ライブラリコール (udp_send.c)

gethostbyname() ライブラリコールは引数で与えられたホスト名に関する hostent 構造体へのポインタを返す．hostent 構造体はヘッダファイル<netdb.h>で以下のように定義されている．

```
struct  hostent {
        char    *h_name;        /* official name of host */
        char    **h_aliases;    /* alias list */
        int     h_addrtype;     /* host address type */
        int     h_length;       /* length of address */
        char    **h_addr_list;  /* list of addresses from name server */
#define h_addr  h_addr_list[0]  /* address, for backward compatibility */
};
```

それぞれのメンバの詳細についてはオンラインマニュアルを参照のこと．ここでは引数のホスト名からその IP アドレス (h_addr) を取得し，構造体 sockaddr_in のメンバ sin_addr に bcopy() を用いて代入している．

2.1.6 bind() システムコール (udp_receive.c)

UNIX のシステムは複数のネットワークインターフェイスを装備することも可能であるため，bind() システムコールを用いてパケットの待ち受けを開始するためには，システムが持つどのインターフェイスを利用するかを指定する必要がある．ソケットのアドレスは INET ドメインでは<netinet/in.h>で定義された構造体 sockaddr_in によって指定する．

```
struct sockaddr_in {
        u_char  sin_len;
        u_char  sin_family;
        u_short sin_port;
        struct  in_addr sin_addr;
        char    sin_zero[8];
};
```

sin_family はアドレスファミリであり，INET ドメインでは AF_INET を指定する．sin_port は接続先のポート番号をネットワークバイトオーダー[6]で指定する．

ここでは，bzero() ライブラリコールを用いて sockaddr_in 構造体を初期化し，svr.sin_addr を INADDR_ANY (すべてのネットワークインターフェイスからパケットを受信する)，svr.sin_port を待ち受けするポート番号 10000 に設定している．sin_zero は使用されない．

[6] **バイトオーダー (byte order)** とは 2 バイト以上の数値をどの順番で (上位バイトから下位バイトの順，またはその逆順など) 読み書きするかを定めたもので，一般に計算機の機種 (CPU) 毎に異なる．しかし，ネットワークを介して異なる機種の計算機が 2 バイト以上の数値を正しくやり取りするためには，ネットワークにおけるバイトオーダーを統一しておく必要がある．これを**ネットワークバイトオーダー (network byte order)**，それに対して各計算機固有のバイトオーダーを**ホストバイトオーダー (host byte order)** という．ソケットアドレスのポート番号はネットワークバイトオーダーで指定しなければならない．htons() 関数によって short 型 (2 バイト整数) の数値をホストバイトオーダーからネットワークバイトオーダーに変換できる．

2.1.7 sendto() システムコール (udp_send.c)

UDP パケットの送出は sendto() システムコールを用いて行う．sendto() は 6 つの引数を取る．1 つ目が使用するパケット，2 つ目が送出するデータ，3 つ目がデータの長さである．ここではデータとして，プログラムに渡された 2 個目の引数と，その文字列としての長さを与えている．4 つ目は送出の詳細を指示するフラグだが今回は特に指定しないため 0 としている．5，6 つ目の引数が宛先を示す `sockaddr` 構造体へのポインタと，構造体のサイズになる[7] ．`host.sin_port` に宛先ポートである 10000 を指定し，gethostbyname() で得た宛先のアドレスから bcopy を用いて `sin_family` 構造体にアドレスをコピーし，これを指定する．

2.1.8 recvfrom() システムコール (udp_receive.c)

UDP パケットの受信は recvfrom() システムコールを用いて行う．`sockaddr_in` 構造体を用いて，どのアドレスからのパケットを受信するかを指定する．ここでは任意のアドレスを受け入れるため，構造体全体を bzero で初期化することで 0.0.0.0 を指定している．recvfrom は 6 つの引数を取り，1 つ目が受信に利用するソケット，2 つ目が受信するバッファ，3 つ目が用意されているバッファのサイズである．ここでは，バッファのサイズを `sizeof(buf)` として recvfrom を呼び出している．C 言語では，データ (バイト配列) を扱う場合，そのデータがどれだけの長さを持つかを別途指定する必要がある．例えばこの場合は，バッファの長さを `sizeof(buf)` で指定し，recvfrom により受信されてバッファに書き込まれたデータの長さを len に格納している．一方，文字列を扱う関数では，このようにデータの長さを別途指定する代わりに，文字列の最後に `'\0'` を付けることでデータ (文字列) の長さが分かるようにしてある[8] ．ここでは，write() システムコール[9] を用いて受信したデータを標準出力に書き出している．残る 2 つの引数は，受け入れるパケットの送信元の指定，兼，受信したパケットの送信元を受け取る引数である．recvfrom が実行された場合は，sender に送信元のアドレスが記録されるため，これを `inet_ntop` ライブラリコールを用いて読みやすい形に変換して，printf で表示している．

2.2 TCP クライアントプログラム

TCP 通信でデータを受け取るプログラムの大まかな流れは以下の通りである．

1. socket() で通信の出入口となるソケットを生成する．
2. connect() で host と接続する．
3. write() でソケットを介して データを送出する．

---

[7] 利用するネットワークプロトコルによってアドレスの表し方はまちまちなため，sockaddr 構造体は内部に sin_family として構造体に格納されたアドレスの種類を持っており，また，引数としてその構造体のサイズを示している．

[8] “すべてのビットが 0” など，通常の文字コードが割り当てられておらず文字列中には出現しないビットパターンが存在するため，このような “このビットパターンが出現するところまでが文字列” という指定の仕方が可能．文字列以外の一般的なデータにはあらゆるビットパターンが出現しうるので，このような終端の指定が不可能なため，別途，サイズを指定せざるを得ない．

[9] write() の第 1 引数 (ファイルディスクプリタ (file descriptor)) に 1 を指定すると標準出力へ出力できる．ちなみに 0 は標準入力，2 は標準エラー出力にあらかじめ割り当てられている．socket() で返される整数もファイルディスクプリタの一種である．

4. read() でソケットを介して データを受信する．

5. write() などで標準出力に受け取った文字列を表示する．

6. close() システムコールで通信を終了する．

UDP との違いは，connect システムコールで接続を確立する必要がある点と，送受信に用いるシステムコールの違いである．

2.2.1 接続の確立

connect() システムコールに対して，ソケット (sock)，接続先のソケットアドレス (host)，およびソケットアドレスの大きさ (sizeof(host)) を与えることにより，接続を確立する．与えるソケットアドレスは，udp_send.c の場合と同様である．

2.2.2 通信

write()，及び，read() システムコールによって，通信 (送受信) を行う．送受信データは第 2 引数で指定した配列 rbuf へ格納する．第 3 引数には配列のサイズを指定する．read() の返り値は実際に受信したバイト数である．ループを用いて read() を繰り返すことで，接続先から送られたデータを受信し続けられる．通信が終了したら close() システムコールによってソケットを閉じる．

3 TCP サーバプログラムの作成

一般にネットワークにおいてサービス (http あるいは telnet,ftp など) を提供する側のプログラムを**サーバプログラム**，サービスを利用する側のプログラムを**クライアント**プログラムと呼ぶ．前節までに解説したのはクライアント側のプログラムの作成法であった．本節ではサーバプログラムの作成法を解説する．

3.1 サーバ側のプログラム例

サーバプログラムは以下の手順に従って通信を行う．

1. socket() で通信の出入口となるソケットを生成する．

2. bind() でソケットとポートを対応づける．クライアントはここで対応づけたポートを介してサーバプログラムと接続する．

3. listen() で最大接続待ち数を指定する．これはサーバがあるクライアントからの接続要求を処理している間，次の接続要求を最大いくつ待たせておくかを指定する数である．

4. accept() でクライアントからの接続要求を待ち，到着したら新たにソケットを生成する．

5. 上で新たに生成したソケットを用いて，write() や read() でクライアントとメッセージの送受信を行う．

6. close() で新たに生成したソケットを閉じる．

3.2 サンプルプログラム — echoserver.c

以降ではこのサンプルプログラムリストに基づいて解説を行う.

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <unistd.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>

int main(int argc,char **argv) {
  int sock,csock;
  struct sockaddr_in svr;
  struct sockaddr_in clt;
  struct hostent *cp;
  int clen;
  char rbuf[1024];
  int nbytes;
  int reuse;

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

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

  /* client 受付用ソケットの情報設定 */
  bzero(&svr,sizeof(svr));
  svr.sin_family=AF_INET;
  svr.sin_addr.s_addr=htonl(INADDR_ANY);
                          /* 受付側の IP アドレスは任意 */
  svr.sin_port=htons(10120);  /* ポート番号 10120 を介して受け付ける */
```

```
  /* ソケットにソケットアドレスを割り当てる */
  if(bind(sock,(struct sockaddr *)&svr,sizeof(svr))<0) {
    perror("bind");
    exit(1);
  }

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

  do {
    /* クライアントの受付 */
    clen = sizeof(clt);
    if ( ( csock = accept(sock,(struct sockaddr *)&clt,&clen) ) <0 ) {
      perror("accept");
      exit(2);
    }

    /* クライアントのホスト情報の取得 */
    cp = gethostbyaddr((char *)&clt.sin_addr,sizeof(struct in_addr),
                       AF_INET);
    printf("[%s]\n",cp->h_name);

    do {
      /* クライアントからのメッセージ受信 */
      if ( ( nbytes = read(csock,rbuf,sizeof(rbuf)) ) < 0) {
        perror("read");
      } else {
        write(csock,rbuf,nbytes);
                    /* 受信文字列をそのままクライアントへ返す (echo) */
      }
    } while ( nbytes != 0 );
          /* read() が 0 を返すまで (=End-Of-File) 繰り返す */

    close(csock);
    printf("closed\n");
  } while(1);  /* 次の接続要求を繰り返し受け付ける */
}
```

3.2.1 ソケットの生成

ソケットの生成はクライアントの場合 (esock プログラム) と同様である．

3.2.2 ソケットアドレス再利用の指定

サーバが異常終了した場合にすぐ再起動できるようにするためのおまじないである．本来ならデバッグが終わった段階で取り除くべきであるが，この演習では必ず残しておくこと．

3.2.3 クライアント受付用ソケットの情報設定

アドレスファミリは `AF_INET`，IP アドレスはどのインタフェースからも受け付けられるように `INADDR_ANY`[10]，ポート番号はここでは 10120 を用いる．

3.2.4 ソケットにアドレスを割り当てる

bind() システムコールは，サーバプログラムがシステムに「これが私のアドレスなので，このアドレス宛に届いたメッセージは私に渡すようにしてください」と言うために使用される．

3.2.5 待ち受けクライアント数の設定

listen() システムコールを用いて

- 接続要求を受け入れる用意があること
- 次の accept() 処理中に到着した接続要求をいくつまで待たせることができるか

を設定する．通常，待ち数 (第 2 引数) はシステムの最大値 5 に設定される．

3.2.6 クライアントの受付

accept() システムコールを用いてクライアントからの connect() による接続要求を受け付ける．受け付けられた場合には受け付けたクライアントと通信するためのソケットを新たに生成し，同時にクライアントのソケットアドレスを取得する．

3.2.7 クライアントのホスト情報の取得

gethostbyaddr() ライブラリコールを用いて接続したクライアントのホスト情報を取得する[11]．このサンプルプログラムでは得られたホスト名を標準出力に出力している．

---

[10] ホスト計算機は一般に複数のネットワークインタフェースを持ち，それぞれに IP アドレスが割り振られている．サーバとクライアントのインターネットにおける位置関係によって，そのクライアントからの接続要求を受信するネットワークインタフェースが一般に変わってくる．通常は接続要求がどこからやってきても受け付けたいので，クライアント受付用 IP アドレスとしては，どのインタフェースの IP アドレスでもよいことを表す `INADDR_ANY` を用いる．

[11] クライアントの情報が特に必要なければ，この処理は省略できる．

3.2.8 クライアントとのメッセージのやり取り

read() を用いてクライアントからメッセージを受信し，それを write() を用いてそのままクライアントに送信する．EOF(End-Of-File) を受信すると，そのクライアントとの接続を切断し，accept() によって次の接続要求を処理する．

3.3 telnet コマンドによる TCP サーバの動作確認

クライアントとサーバがどちらも自作プログラムでうまく動作しない場合，クライアントとサーバのどちらに問題があるのかを特定しにくい．今回のように TCP 通信でサーバがポートを開放して接続を待ち受ける場合，telnet コマンドを使ってサーバが思い通りに動作しているかを以下の手順で確認すると良い[12]．

まず，あらかじめ適当なホスト (例えば exp101) にリモートログイン（“ssh exp101”）して “echoserver” を実行しておく．次に，自分の使っている端末 (別の kterm) で以下のように telnet コマンドを入力する．

```
% telnet exp101 10120
Trying 192.168.16.101...
Connected to exp101.exp.ics.es.osaka-u.ac.jp.
Escape character is '^]'.
```

exp101 で動作しているサーバがポート番号 10120 で TCP ソケット通信を待ち受けていれば，上記のように入力を受け付けるモードになる．この段階で失敗するようであれば，サーバのアドレスかポート番号が間違っている可能性が高い．ここで例えば”hello” と入力し，エンターキーを押下すると，echoserver が正しく実装されていれば，以下のように入力した文字列と同じ文字列が出力される．

```
hello
hello
```

1 行目の”hello” は入力した文字列で，2 行目の”hello” は echoserver から送られてきた文字列である．telnet を終了させる時は，以下のように Ctrl-] を入力して telnet 自身のコマンドを受け付けるモードに切り換えた後，close コマンドを入力すれば良い．

```
^]
telnet> close
Connection closed.
%
```

[12]telnet は，本来 telnet サーバとの間でポート番号 23 を使った TCP ソケット通信により，単純なテキストの送受信を行うコマンドであるが，ポート番号を指定して実行すると指定したポートを使って TCP ソケット通信を行うことができる．

4　課題 2-1

echoserver プログラムと通信する以下のような**クライアント**プログラムを作成せよ．

echoclient host

echoclient プログラムは host で動作している echoserver プログラムとポート番号 10120 を介して接続し，標準入力から読み込んだ文字列を echoserver プログラムへ送信し，受信した文字列を標準出力に書き出す．これを標準入力から EOF を受け取るまで (標準入力が端末なら Ctrl-D を入力するまで) 繰り返す．

動作確認は次の手順で行うとよい．

1. 3.3 節の手順に従い，適当なホスト (ここでは exp101 とする) にリモートログインして echoserver を実行し，telnet コマンドを使って echoserver が正しく動作していることを確認する．
2. 自分の使っている端末 (別の kterm) で “echoclient exp101” を実行する．exp101 は実際に echoserver を動かしているホスト名に変えること．
3. 端末から適当な文字列を入力してみる．改行で同じ文字列が表示されればよい．終了は Ctrl-D.

なお，他の人がすでに echoserver を動かしているホストでは echoserver を実行できない．これは同じホストの同じポート番号で待ち受け出来るのは一つのプロセスに限られるからである．このような場合，ホストを変えるかポート番号を (サーバ，クライアント両方ともに) 隣接する他の番号に変えてみるとよい．ポート番号は 1024 以上で他のプロセスが使っていない番号であればなんでもよいが，/etc/services に記述されている番号は避けること．

5　全二重通信を行うプログラム — 簡易 talk

UNIX には他のユーザと通信するための **talk** というプログラムがある．talk は，端末を用いた 1 対 1 の通信を提供している．つまり，一方のユーザの端末から入力された文字列を他方のユーザの端末上にコピーするのである．

UNIX における talk プログラムは，実用的に使えるようにさまざまな工夫がなされているが，本課題では talk の必要最小限の機能のみを実現するプログラム，**簡易 talk** を作成する．

5.1　簡易 talk の仕様

自分 (me) は計算機 host1 にログインしており，通信したい相手 (you) が計算機 host2 にログインしているとする．このとき，簡易 talk の通信は次のようにして確立するものとする．

1. me はサーバプログラム simple-talk-server を次のようにして起動する．
   ```
   % simple-talk-server
   ```
2. you は (me と話がしたければ)host 2 上で次のようにしてクライアントプログラム simple-talk-client を起動する．
   ```
   % simple-talk-client host1
   ```

3. simple-talk-server, simple-talk-client はそれぞれ host1, host2 の端末上に “connected” とメッセージを出して通信が確立したことを me と you に知らせる．

通信が確立したら，一方のユーザの端末から入力された文字列を他方のユーザの端末上にコピーする．

簡易 talk の通信に用いるポート番号は **10130** とする．

5.2 簡易 talk の試作 — simple-talk-server.c

簡易 talk は，受信文字列をそのままコピーするという点で，課題 2 前半で学んだ echoserver プログラムに良く似ている．echoserver と simple-talk-server で異なるのは以下の 2 点である．一つは，コピーする先が異なる点である．echoserver は受信した文字列をクライアントにそのまま返していたのだが，簡易 talk では受信した文字列をクライアントの話相手であるサーバ自身の端末にコピーする[13]．

もう一つは，対称な双方向通信であるという点である．echoserver の場合，クライアントから一方的に文字列を受け取りそれを返すだけであったが，簡易 talk では並行してサーバ側からも端末から文字列を入力して相手（クライアント）の端末にコピーする．したがって，サーバ側，クライアント側ともに端末とソケットの両方から同時に入力を受け付けなければならない[14]．これを入力の**多重化** (multiplexing) という．入力の多重化の方法には少なくとも次の 3 つがある．

1. 複数のプロセスによる多重化

   UNIX はマルチタスクの OS であるために，端末からの入力とソケットからの入力とを別のプロセスやスレッドに扱わせることで多重化が行える．

2. ノンブロッキング I/O を使用する．

   read() システムコールがブロック (=データが来るまで停止して待つこと) しないように設定し，実際に読み出し動作を行うことによってデータの到着を知る．

3. `select()` によってデータの到着を監視する．

   `select()` システムコールによって，一般にファイルディスクリプタ (ソケット，標準入出力，端末，ファイルなど) のうちどれが読取りに応じられるかを知ることができる．

本課題では，上記の 3 つの中で最も処理が簡単な 3. の `select()` システムコールを使用する．

5.3 select() システムコールについて

select() システムコールは，一度に複数のファイルディスクリプタ (file descriptor) の状態の変化を監視する．具体的には，読み込みが可能になったか，書き込みが可能になったか，あるいは，例外 (帯域外受信) が発生したかを監視できる．ファイルディスクリプタとしてはソケット以外に

[13] simple-talk-server はユーザからの端末入出力も処理するという点で，厳密な意味でのサーバとはいえないが，ここでは相手からの接続要求を待つ側という意味でサーバという言葉を用いている．

[14] 必ず交互に発言する，と言った取り決めを行っておけば，このようなプログラム上の改良は必要ではないが，非常に使いにくいアプリケーションとなってしまう．

標準入出力，open() システムコールでオープンした通常のファイルやデバイス (シリアルポートなど) も指定可能である．

select() の第 1 引数は監視するファイルディスクリプタの最大値に 1 を加えた整数を指定する．第 2〜4 引数はそれぞれ，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を指定する．監視しない項目は NULL ポインタを指定する．ファイルディスクリプタの集合を表すデータ型としては，fd_set という型がヘッダファイル<sys/types.h>で定義されている．さらにその集合を操作するマクロとして，FD_ZERO(), FD_SET(), FD_ISSET() などが定義されている．FD_ZERO() は集合を空にするマクロで，FD_SET() は集合にファイルディスクリプタを追加するマクロ，FD_ISSET() は集合に指定したファイルディスクリプタが入っているか調べるマクロである．第 5 引数は select() を監視する待ち時間を構造体 struct timeval へのポインタによって指定する．struct timeval はヘッダファイル<sys/time.h>で定義されている．待ち時間として 0 秒を指定するとその瞬間の状態を調べて select() は直ちに終了する．また，第 5 引数に NULL ポインタを指定すると，監視するファイルディスクリプタのいずれかの状態が変化するまで無期限にブロック (停止) する．select() が終了すると，状態が変化したファイルディスクリプタの集合を，第 2〜4 引数で指定されたポインタが指す場所に格納し，返り値として，状態が変化したファイルディスクリプタの総数，あるいは-1(エラーの時) を返す．

下記に select を用いた基本的なプログラム例を示す．ポイントとなる部分以外は「...」にて省略している．

```
int csock;            /* クライアントを受け付けたソケット */
fd_set rfds;          /* select() で用いるファイル記述子集合 */
struct timeval tv;    /* select() が返ってくるまでの待ち時間を指定する変数 */
...
/* クライアントからの接続待ち受けなど */
...
do{
  /* 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 */
  FD_ZERO(&rfds);      /* rfds を空集合に初期化 */
  FD_SET(0,&rfds);     /* 標準入力 */
  FD_SET(csock,&rfds);  /* クライアントを受け付けたソケット */

  /* 監視する待ち時間を 1 秒に設定 */
  tv.tv_sec = 1;
  tv.tv_usec = 0;

  /* 標準入力とソケットからの受信を同時に監視する */
  if(select(csock+1,&rfds,NULL,NULL,&tv)>0) {
    if(FD_ISSET(0,&rfds)) {  /* 標準入力から入力があったなら */
      /* 標準入力から読み込みクライアントに送信 */
      ...
    }
```

```
    if(FD_ISSET(csock,&rfds)) { /* ソケットから受信したなら */
      /* ソケットから読み込み端末に出力 */
      ...
    }
  }
} while(1);  /* 繰り返す */
```

ここでは，標準入力 (ファイルディスクリプタは 0) およびクライアントと接続したソケット (csock) の 2 つがそれぞれ読み込み可能になったかどうかを 1 秒間ずつ監視している．そして，select() が終了して読み込み可能になったファイルディスクリプタが存在するならば，それらに対して入出力処理を行っている．複数のソケットからの待ち受けを行う場合は，`FD_SET()` を各ソケットに対して実行すれば良い．その際，select() を呼び出す際に指定する監視するファイルディスクリプタの最大値+1 は，プログラム中で値を求めておく必要がある．

6 課題 2-2

本課題の手引きを参考に，simple-talk-server プログラムを完成させよ．また，サーバ側と同様に select() システムコールを用いて，クライアント側のプログラム simple-talk-client を作成せよ．最低限，以下の仕様を満たすプログラムとすること．

1. simple-talk-server を起動してから，simple-talk-server が動作しているホスト名を指定して，simple-talk-client を起動することで接続が確立する．
2. 接続確立後は，simple-talk-server 側で標準入力から入力した文字列が**即座に** simple-talk-client 側に表示される．逆も同様．
3. 全二重通信で実現すること．すなわち，会話をする 2 人がどのような順序で発言を行っても，相手側に即座にそのメッセージが表示されること．

なお，以下のような問題が見られる場合は，全二重通信を正しく実装できていない可能性が高い[15]．

- 交互に発言しなければメッセージが伝わらない．
- 相手がメッセージを送信してきても，それらが即座には表示されない．自分が発言しようとリターンキーを押した瞬間に，それまでに相手が送ったメッセージがまとめて表示される．

それ以外の仕様に関しては各自で定め，どのようなプログラムにしたかをレポートに記載すること．

7 発展課題

余力がある者は以下の課題に挑戦してみよ．

1. echoserver を改造して，受信した文字列中のすべての大文字を小文字に変換して返すサーバ lowerechoserver を作成せよ．他の変換を考えても良い．

[15] 他の理由のバグである可能性ももちろん存在する．

2. 例示したプログラムの断片はエラー処理などに関しては考慮していない．どのような例外的な動作が起こりうるかを考え，それらに対するエラー処理を実装せよ．例えば，simple-talk-server の場合，例外的な事例として，1 つのサーバに複数のクライアントが接続しようとした場合などが考えられる．

3. simple-talk-server,simple-talk-client プログラムに手を加えて，サーバ側，クライアント側それぞれの発言者の名前を登録し，各行に誰の発言であるか表示する機能を追加せよ．その他，便利と思われる機能を自由に追加してよい．

4. 2 つのクライアント A,B と接続し，クライアント A から受信した文字列をクライアント B へ送るサーバ twoclientserver を作成せよ．

5. UDP では，TCP とは異なり，パケットのブロードキャストが行える．演習室などの IPv4 の環境では，UDP パケットの宛先を 255.255.255.255 とした場合，通信可能なすべてのホストでそのパケットが受信される[16] ．ただしこの機能は送信を行うソケットに対して特殊な設定をしないと利用できない．教科書やインターネット上の情報を探して，udp_sender をブロードキャストが利用できるように改造し，動作を確かめてみよ．

6. UDP の場合も select() システムコールは同様に利用可能である．UDP を用いる simple-talk-udp を作成せよ．この場合，-server と-client を別々に作成する必要は必ずしも無い．会話する両者共に，(1) 互いのアドレスを引数として起動する，(2) 特定のポート番号で UDP パケットを待ち受け，メッセージを表示する，(3) 標準入力からの入力を指定されたアドレスへ UDP パケットとして送信する，の 3 つの機能を持たせるだけで実現できる．以下に基本的な手順を例示する．

   (a) socket() システムコールで INET ドメイン，データグラム型ソケットを生成する．(`socket(AF_INET,SOCK_DGRAM,IPPROTO_UDP)`)

   (b) bind() システムコールでソケットに待ち受けアドレスを設定する．

   (c) select() システムコールで，標準入力とソケットからの入力を待ち受ける．

   (d) 標準入力から入力があった場合は送信は sendto() システムコールでメッセージを送信し，ソケットからの入力があった場合は recvfrom() システムコールでメッセージを受信して表示する．詳細は本資料の他，`man 2 sendto` および `man 2 recvfrom` などのコマンドで表示されるマニュアル参照のこと．

   (e) 標準入力が閉じられた場合は close() システムコールでソケットを閉じる．

   ただし，この仕様では，両者が正しく相手のアドレスを指定して起動しない限り会話は成立しない点に注意すること．また，simple-talk-server と simple-talk-client と同様に “connected” とメッセージを表示させるためには，そのためのパケットを明示的に送信する必要がある．同様に，自分がプログラムを終了しても会話の相手にはそのことは伝わらない．このような問題を含め，便利と思われる仕様を策定し，それに応じた機能を追加せよ．

[16] 一般にインターネットはルータと呼ばれる機器で LAN に区切られている．通常，ブロードキャストパケットはルータを超えて転送されないように設定されるため，基本的にブロードキャストは LAN 内すべての機器に対してパケットを送る行為と考えればよい．LAN に繋いだだけで利用可能になる機器などは主にこのような仕組みを用いて，必要なサービスを探索することで実現されている．

7. UDP を用いた代表的なサービスを列挙し，TCP に比べて UDP を用いる利点・欠点を考察せよ．簡易 talk は UDP を用いるのにふさわしいアプリケーションといえるか？

8. 課題 2-1 で作成したプログラムを一部改造することで World Wide Web から情報を取得する簡単なクライアントプログラムが作成できる．HTTP サーバから情報を取得する簡易 HTTP クライアントを作成せよ．現在一般的な HTTP 1.1 を用いてファイルを取得する場合，取得したいファイルの URI が，

   http://host_name/directory_names/file_name

   であるとすると (以下の説明の “host_name”，“directory_names”，“file_name” は，それぞれ取得したいファイルの URI の該当部分に置き換えること)，

   (a) TCP のソケットを生成し，“host_name” のポート番号 80 に接続する
   (b) “GET /directory_names/file_name HTTP/1.1\n” を送信する
   (c) “Host: host_name\n” を送信する
   (d) “Connection: close\n” を送信する
   (e) 改行を送信する
   (f) ファイルの内容 (もしくはエラーメッセージ) がサーバから送られてくるので，それらを受信して表示か保存する

   という手順となる (ただしここで，“\n” は改行文字)．HTTP 1.1 では，通信の条件などを *HTTP* **ヘッダ**を用いてサーバへ送ることが出来る．ヘッダの終わりは，何も書かれていない空行を送信する事で示す仕様となっているため，e の手順が必要となる．詳しくは W3C のページ (http://www.w3c.org/) 等を参照のこと．

   ただし，演習室は外部との直接通信が制限されているため，学科内などの一部のサーバを除いては，上記の方法では通信が出来ない．演習室から外部の HTTP サーバと通信させるには，通信を仲介する Proxy サーバを用いればよい．演習室の Proxy サーバは，“cacheserv.ics.es.osaka-u.ac.jp” で，ポート番号 3128 番でサービスを提供している．これを用いる場合は，上記の手順のうち，a と b を下記の a’，b’ のように変えれば良い．

   (a’) TCP のソケットを生成し，“cacheserv.ics.es.osaka-u.ac.jp” のポート番号 3128 に接続する
   (b’) “GET http://host_name/directory_names/file_name HTTP/1.1\n” を送信する

   c 以降は同様である．Proxy を介する場合には，“GET” の後に “http://host_name” から記述する必要がある事に注意．また，演習室以外で異なる Proxy サーバを使う必要がある場合は接続先を適宜変更すること．

8　レポート提出について

レポートと作成したプログラムのソースコードのファイルを締切日の演習開始前に，CLE へ添付ファイルとして提出すること．レポートの形式は PDF とする．なお，演習開始の時点で提出が終わっていなかった分は提出遅れとして扱う．レポートのスタイルは特に問わないが，TeX の使用を推奨する．レポートの表紙は，1 ページを使って，

- 授業名
- 課題名
- 学籍番号，名前

を書くこと．手書きのレポートは認めない．レポートの内容に関しては，少なくとも，

- 実行結果
- 考察
- 感想，意見，疑問等

を盛り込むこと．もちろんその他にもレポートに書くべきことはあるが，それについては，演習の説明会の時に配布した資料等を参考にして作成すること．

締め切りは，

5/28(月)12:59

とする．