情報システム工学 II(システム開発工学)

ネットワークプログラミング

Network Programming with Sockets

山田泰司

taiji@aihara.co.jp

株式会社あいはら 研究開発チーム

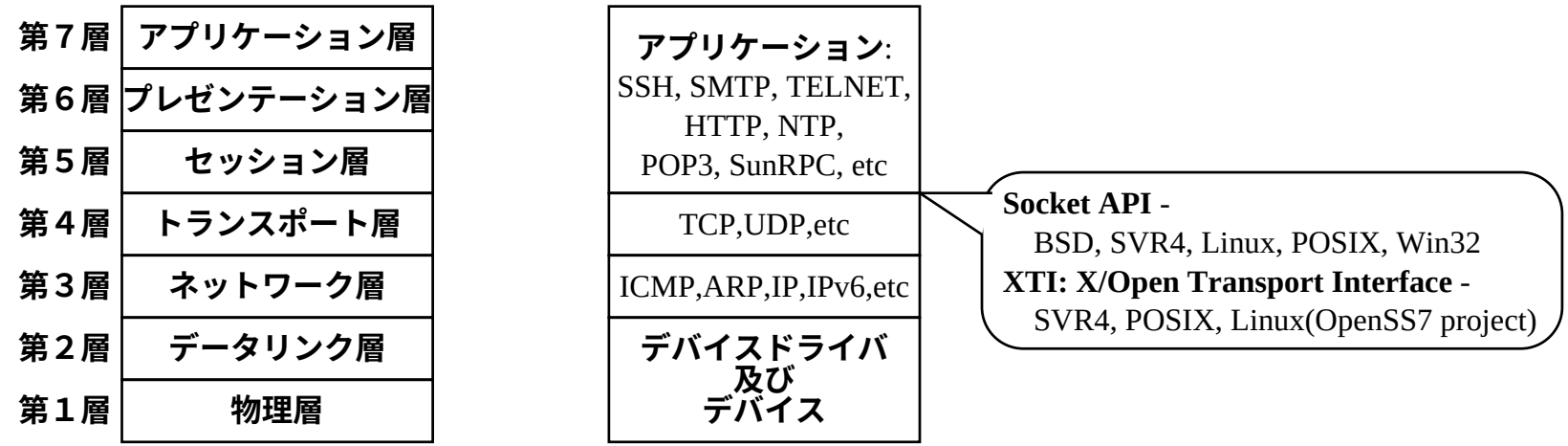
ネットワークプログラミングとは

今や事実上の標準となった BSD ソケット API でプログラミングを行なうのが基本：

- ▷ 4.2BSD 由来／互換のアプリケーションインターフェース Solaris, Linux, etc.
- ▷ MS Windows 向けの実装は WinSock, etc.

対抗するネットワーキング API として XTI: X/Open Transport Interface：

- ▷ SVR3 の TLI:Transport Layer Interface 由来
- ▷ X/Open により XTI へと拡張、標準化、そして POSIX.1g の一部となる
- ▷ SVR4 より TCP/IP をはじめ、さまざまなプロトコルに対応



ソケット API とは

大まかには以下の手順で相手（ピア）と通信を行なう：

1. socket でソケットデスクリプタを取得
2. bind でソケットにローカルなアドレスを結合（歴史的に「ソケットの命名、ソケットに名前を付ける」などと呼ぶ）
3. ストリームサーバは listen でソケットを受動 (passive) 状態へ変換
4. ストリームサーバは accept で接続待ちキュー (connect による接続要求ログ) の先頭について、接続済みソケットデスクリプタとピアのアドレスを取得
5. ストリームクライアントは能動 (active) 状態であるソケットを connect により、ピアのアドレス (の accept) へ接続要求し、ソケットをデータ転送状態へ移行
6. データグラムは recvfrom, sendto でデータを読み書き
7. ストリームは read(もしくは recv), write(もしくは send) でデータを読み書き
8. 後始末として、ソケットデスクリプタに close(もしくは shutdown) を適用

主なソケット API：

```
$ less /usr/include/sys/socket.h       # BSD, Solaris, Linux
        :
int     socket(int family, int type, int protocol);

int     bind(   int sockfd, const struct sockaddr *selfaddr, socklen_t addrlen);

int     listen( int sockfd, int backlog);
int     accept( int sockfd,       struct sockaddr *clntaddr, socklen_t *addrlen);

int     connect(int sockfd, const struct sockaddr *servaddr, socklen_t addrlen);

ssize_t recvfrom(int sockfd,       void *buf, size_t nbytes, int flags,       struct sockaddr *peeraddr, socklen_t *addrlen);
ssize_t sendto(  int sockfd, const void *buf, size_t nbytes, int flags, const struct sockaddr *peeraddr, socklen_t addrlen);

ssize_t recv(int sockfd,       void *buf, size_t nbytes, int flags);
ssize_t send(int sockfd, const void *buf, size_t nbytes, int flags);
        :
```

ここで気になるのは、family、type、protocol、struct sockaddr、socklen_t であろう。

まず片付けておきたいのは、socklen_t である。実は、POSIX 等の規格の制定上において混乱があった事がマニュアルページに書かれており、レガシーなところから int, size_t, uint32_t といろいろである。結局のところ、符号や表現し得る最大値に係る値に障らないようコーディングした方が無難である。

ソケット API とは（つづき）

その他のソケット API：

```
$ less /usr/include/sys/socket.h       # BSD, Solaris, Linux
        :
ssize_t recvmsg(int sockfd,       struct msghdr *, int flags);
ssize_t sendmsg(int sockfd, const struct msghdr *, int flags);

int     shutdown(int sockfd, int howto);

int     getsockopt(int sockfd, int level, int optname,       void *optval, socklen_t *optlen);
int     setsockopt(int sockfd, int level, int optname, const void *optval, socklen_t optlen);

int     getpeername(int sockfd, struct sockaddr *peeraddr, socklen_t *addrlen);
int     getsockname(int sockfd, struct sockaddr *selfaddr, socklen_t *addrlen);

int     socketpair(int family, int type, int protocol, int sockfd[2]);
        :
```

マニュアルページを読むときに、セクションを指定しないと無関係なページが表示される関数があるので注意：

```
$ man -s 2 accept                      # BSD, Linux
$ man -s 3SOCKET accept                # Solaris
```

アドレスファミリ、ソケットタイプほか

アドレスファミリ (family) **に指定できる定数：**

AF_INET	IPv4 **ソケット**
AF_INET6	IPv6 **ソケット**
AF_UNIX	Unix **ドメインソケット** (AF_LOCAL **と同値**)
AF_ROUTE	**内部経路制御ソケット**

ソケットタイプ (type) **に指定できる定数：**

SOCK_STREAM	**信頼性のある接続指向 (順序付き) のストリームソケット**
SOCK_DGRAM	**信頼性のないコネクションレス (順序なし) の固定最大長のあるデータグラムソケット**
SOCK_RAW	**生の** IPv4/IPv6 **ソケット**

protocol **に指定できる定数は、**type==SOCK_RAW **の場合を除き、通常** 0 **を指定。**

入出力関数の flags **に指定できる定数は、基本的には** 0 **を指定。**

listen **の** backlog **に指定できる定数は、通常** 5 **などの** 1 **以上の値であるが、より大きい値を指定できるようにしておくと望ましい。但し、システムによって上限が異なるので注意が必要。**

howto **に指定できる定数：**

SHUT_RD	**それ以降の受信を禁止**
SHUT_WR	**それ以降の送信を禁止**
SHUT_RDWR	**それ以降の送受信を禁止**

ソケットアドレスとは

ソケットアドレスはローカルなアドレス、ピアのアドレスを設定、取得するために使用：

```
$ less /usr/include/sys/socket.h        # BSD
        :
struct sockaddr {
  uint8_t       sa_len;         /* total length */
  sa_family_t   sa_family;      /* address family */
  char          sa_data[14];    /* addr value (actually larger) */
};

$ less /usr/include/sys/socket*.h       # Solaris
$ less /usr/include/bits/socket.h       # Linux
struct sockaddr {
  sa_family_t   sa_family;      /* address family */
  char          sa_data[14];    /* up to 14 bytes of direct address */
};
        :
```

これを「総称ソケットアドレス構造体」という。

ソケットアドレス構造体とは

「総称ソケットアドレス構造体」の実体は「IPv4 ソケットアドレス構造体」「IPv6 ソケットアドレス構造体」「Unix ドメインソケット構造体」など、さまざまなアドレスファミリに依存

IPv4 **ソケットアドレス構造体：**

```
$ less /usr/include/netinet/in.h          # BSD
struct sockaddr_in {
  uint8_t               sin_len;          /* total length */
  sa_family_t           sin_family;       /* AF_INET */
  in_port_t             sin_port;         /* TL port # */
  struct in_addr        sin_addr;         /* IP address */
  char                  sin_zero[8];      /* Padding */
};

$ less /usr/include/netinet/in.h          # Solaris, Linux
struct sockaddr_in {
  sa_family_t           sin_family;       /* AF_INET */
  in_port_t             sin_port;         /* TL port # */
  struct  in_addr       sin_addr;         /* IP address */
  unsigned char         sin_zero[8];      /* Padding */
};
```

ポート番号 (sin_port) **は** 0〜65535、**但し** 0〜1023 **は特権付きポート、**1024〜49151 **は予約済みポートであるので、通常は** 49152〜65535 **を用いること。**

ソケットアドレス構造体とは（つづき）

IPv6 **ソケットアドレス構造体：**

```
$ less /usr/include/netinet6/in6.h      # BSD
struct sockaddr_in6 {
  uint8_t               sin6_len;       /* total length */
  sa_family_t           sin6_family;    /* AF_INET6 */
  in_port_t             sin6_port;      /* TL port # */
  uint32_t              sin6_flowinfo;  /* IPv6 flow information */
  struct in6_addr       sin6_addr;      /* IPv6 address */
  uint32_t              sin6_scope_id;  /* scope zone index */
};

$ less /usr/include/netinet/in.h        # Solaris, Linux
struct sockaddr_in6 {
  sa_family_t           sin6_family;    /* AF_INET6 */
  in_port_t             sin6_port;      /* TL port # */
  uint32_t              sin6_flowinfo;  /* IPv6 flow information */
  struct in6_addr       sin6_addr;      /* IPv6 address */
  uint32_t              sin6_scope_id;  /* Depends on IPv6 scope */
        :
};
```

ポート番号 (sin6_port) **は** 0〜65535**、但し** 0〜1023 **は特権付きポート、**1024〜49151 **は予約済みポートであるので、通常は** 49152〜65535 **を用いること。**

ソケットアドレス構造体とは（つづき）

Unix **ドメインソケット構造体：**

```
$ less /usr/include/sys/un.h              # BSD
struct sockaddr_un {
  unsigned char         sun_len;          /* sockaddr len including null */
  sa_family_t           sun_family;       /* AF_UNIX */
  char                  sun_path[104];    /* path name (gag) */
};

$ less /usr/include/sys/un.h              # Solaris, Linux
struct sockaddr_un {
  sa_family_t           sun_family;       /* AF_UNIX */
  char                  sun_path[108];    /* path name (gag) */
};
```

Unix **ドメインソケットパス** (sun_path) **は** NULL **終端されたファイルシステム上のパス名を指定する。セキュリティ上、作成する場所やパーミッションについては十分に配慮すべきであるが、ここではローカルに悪意ある者がいないものと仮定して、単に**/tmp **配下に作成するものとする。**

ソケット family, type の違い

```
$ less /usr/include/sys/socket.h        # BSD, Solaris
$ less /usr/include/bits/socket.h       # Linux
        :
#define AF_UNSPEC       0       /* unspecified */
#define AF_UNIX         1       /* local to host (pipes) */
#define AF_LOCAL        AF_UNIX /* backward compatibility */
#define AF_INET         2       /* internetwork: UDP, TCP, etc. */
        :

$ less /usr/include/sys/socket.h        # OpenBSD
        :
#define AF_INET6        30      /* IPv6 */
        :

$ less /usr/include/sys/socket.h        # Darwin
        :
#define AF_INET6        24      /* IPv6 */
        :

$ less /usr/include/sys/socket.h        # Solaris
        :
#define AF_INET6        26      /* IPv6 */
        :

$ less /usr/include/bits/socket.h       # Linux
        :
#define PF_INET6        10      /* IPv6 */
        :
```

このようにある定義まではどのプラットフォームでも数値が等しいが、BSD 系である Darwin(Mac OS X) でさえ、BSD の値と値が異なるなど、システムによって数値が異なるので、プログラムでは決して数値では指定しないこと。

ソケット family, type の違い（つづき）

```
$ less /usr/include/sys/socket.h        # BSD
        :
#define SOCK_STREAM     1       /* stream socket */
#define SOCK_DGRAM      2       /* datagram socket */
#define SOCK_RAW        3       /* raw-protocol interface */
#define SOCK_RDM        4       /* reliably-delivered message */
#define SOCK_SEQPACKET  5       /* sequenced packet stream */
        :

$ less /usr/include/sys/socket.h        # Solaris
        :
#define SOCK_STREAM     2       /* stream socket */
#define SOCK_DGRAM      1       /* datagram socket */
#define SOCK_RAW        4       /* raw-protocol interface */
#define SOCK_RDM        5       /* reliably-delivered message */
#define SOCK_SEQPACKET  6       /* sequenced packet stream */
        :

$ less /usr/include/bits/socket.h       # Linux
        :
#define SOCK_STREAM     1       /* Sequenced, reliable, connection-based byte streams */
#define SOCK_DGRAM      2       /* Connectionless, unreliable datagrams of fixed maximum length */
#define SOCK_RAW        3       /* Raw protocol interface */
#define SOCK_RDM        4       /* Reliably-delivered messages */
#define SOCK_SEQPACKET  5       /* Sequenced, reliable, connection-based, datagrams of fixed maximum length */
        :
```

Linux では 4.2BSD 由来を重んじて BSD 系と同値となっているが、SVR4 系の Solaris では異なる値をもつ。よって、こちらもやはりプログラムでは決して数値では指定しないこと。

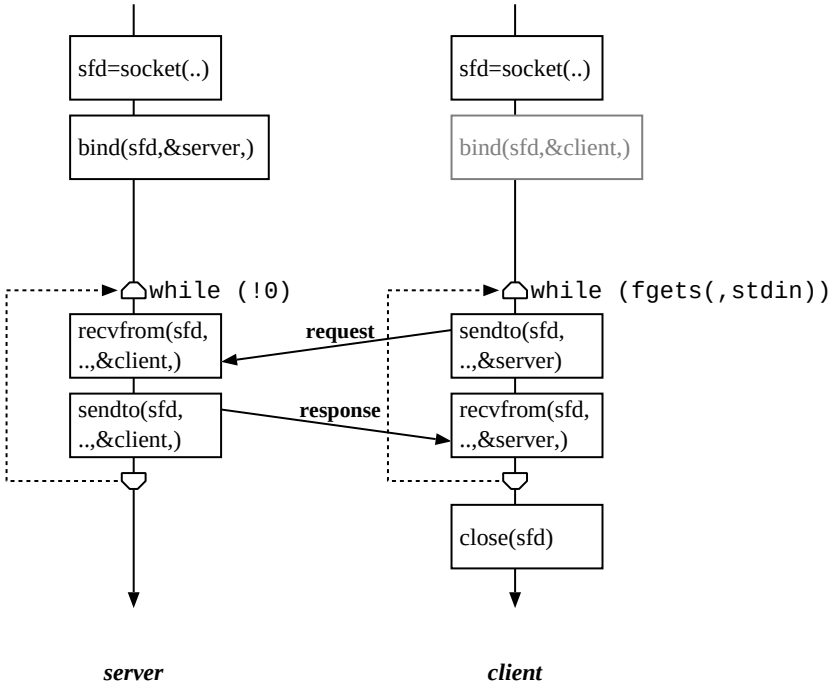
（例題１）各種echoサーバ／クライアントの作成

各種 echo **サーバ／クライアントを、**Unix **ドメイン（**AF_UNIX**）、**IPv4**（**AF_INET**）、**IPv4/IPv6 **両用（**AF_INET/AF_INET6**）それぞれのアドレスファミリ（**family**）について、データグラム（**SOCK_DGRAM**）、ストリーム（**SOCK_STREAM**）それぞれのソケットタイプ（**type**）について作成せよ。**

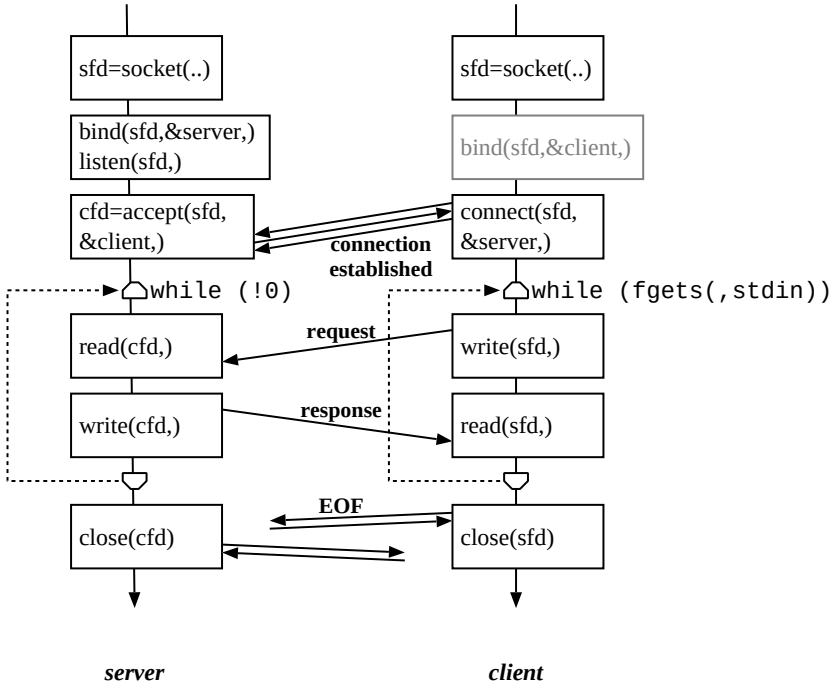
また、Unix **ドメイン以外については、接続相手のアドレスを表示するようにせよ。**

サーバ／クライアント ファイル名

type ＼ family	AF_UNIX	AF_INET	AF_INET/AF_INET6
SOCK_DGRAM	sund-echod/sund-echoc	sind-echod/sind-echoc	sand-echod/sand-echoc
SOCK_STREAM	suns-echod/suns-echoc	sins-echod/sins-echoc	sans-echod/sans-echoc



SOCK_DGRAM: UDP/INET, UDP/INET6, UDP/UNIX



SOCK_STREAM: TCP/INET, TCP/INET6, TCP/UNIX

（例題１）の解答例：データグラム

sockets/sund-echod.c: AF_UNIX, SOCK_DGRAM, echo **サーバ**

```
        :
#include <sys/types.h>  /* sys/socket.h, etc */
#include <sys/socket.h> /* socket, etc */
#include <unistd.h>     /* read, write, close, etc */
#include <sys/un.h>     /* struct sockaddr_un, etc */
        :
int main(int argc, char *argv[])
{
  int sfd;
  struct sockaddr_un server, client;
  socklen_t addr_size;
  char *sun_path = "/tmp/sund-echod.socket", buf[1024];
        :
  if ((sfd = socket(AF_UNIX, SOCK_DGRAM, 0)) == -1) { perror("server: socket"); exit(1); }
  memset(&server, 0, sizeof(server));
  server.sun_family = AF_UNIX;
  strncpy(server.sun_path, sun_path, sizeof(server.sun_path)-1);
  unlink(server.sun_path);
  if (bind(sfd, (struct sockaddr *)&server, sizeof(server)) == -1) { perror("server: bind"); exit(1); }
  while (!0) {
    addr_size = sizeof(client);
    if (recvfrom(sfd, buf, sizeof(buf), 0, (struct sockaddr *)&client, &addr_size) == -1) { perror("server: recvfrom"); exit(1); }
    if (sendto(sfd, buf, sizeof(buf), 0, (struct sockaddr *)&client, addr_size) == -1) { perror("server: sendto"); exit(1); }
  }
  close(sfd);
  unlink(sun_path);
  return 0;
}
```

unlink(server.sun_path) **は、つづく** bind **に失敗しないよう、ゴミとして残っているかも知れないソケットファイルを削除している。**

（例題１）の解答例（つづき）

sockets/sund-echoc.c: AF_UNIX, SOCK_DGRAM, echo **クライアント**

```
        :
#include <sys/types.h>  /* sys/socket.h, etc */
#include <sys/socket.h> /* socket, etc */
#include <unistd.h>     /* read, write, close, etc */
#include <sys/un.h>     /* struct sockaddr_un, etc */
        :
int main(int argc, char *argv[])
{
  int sfd;
  struct sockaddr_un server, client;
  socklen_t addr_size;
  char *sun_path = "/tmp/sund-echod.socket", *sun_temp = tempnam("/tmp", "sund-echoc.socket-"), buf[1024];
        :
  if ((sfd = socket(AF_UNIX, SOCK_DGRAM, 0)) == -1) { perror("client: socket"); exit(1); }
  memset(&client, 0, sizeof(client));
  client.sun_family = AF_UNIX;
  strncpy(client.sun_path, sun_temp, sizeof(client.sun_path)-1);
  if (bind(sfd, (struct sockaddr *)&client, sizeof(client)) == -1) { perror("client: bind"); exit(1); }
  memset(&server, 0, sizeof(server));
  server.sun_family = AF_UNIX;
  strncpy(server.sun_path, sun_path, sizeof(server.sun_path)-1);
  while (fgets(buf, sizeof(buf), stdin)) {
    if (sendto(sfd, buf, sizeof(buf), 0, (struct sockaddr *)&server, sizeof(server)) == -1) { perror("client; sendto"); exit(1); }
    addr_size = sizeof(server);
    if (recvfrom(sfd, buf, sizeof(buf), 0, (struct sockaddr *)&server, &addr_size) == -1) { perror("client: recvfrom"); exit(1); }
    printf("%s", buf);
  }
  close(sfd);
  return 0;
}
```

AF_UNIX, SOCK_DGRAM **クライアントでは、サーバが応答を返すためのソケットを** bind **しておかなければならない。パス名はサーバにとって未知で構わないので、ここでは** tempnam **によりユニークな一時ファイルを用いている。**

（例題１）の解答例（つづき）

sockets/sind-echod00.c: AF_INET, SOCK_DGRAM, echo **サーバ**

```
        :
#include <sys/types.h>  /* sys/socket.h, etc */
#include <sys/socket.h> /* socket, etc */
#include <unistd.h>     /* read, write, close, etc */
#include <netinet/in.h> /* struct sockaddr_in, etc */
        :
int main(int argc, char *argv[])
{
  int sfd;
  struct sockaddr_in server, client;
  socklen_t addr_size;
  in_port_t port = 50000;
  char buf[1024];
        :
  if ((sfd = socket(AF_INET, SOCK_DGRAM, 0)) == -1) { perror("server: socket"); exit(1); }
  memset(&server, 0, sizeof(server));
  server.sin_family = AF_INET;
  server.sin_port = htons(port);
  server.sin_addr.s_addr = htonl(INADDR_ANY);
  if (bind(sfd, (struct sockaddr *)&server, sizeof(server)) == -1) { perror("server: bind"); exit(1); }
  while (!0) {
    addr_size = sizeof(client);
    if (recvfrom(sfd, buf, sizeof(buf), 0, (struct sockaddr *)&client, &addr_size) == -1) { perror("server: recvfrom"); exit(1); }
    if (sendto(sfd, buf, sizeof(buf), 0, (struct sockaddr *)&client, addr_size) == -1) { perror("server: sendto"); exit(1); }
  }
  close(sfd);
  return 0;
}
```

htons(port) は port=50000 がホストバイトオーダであるため、ネットワークバイトオーダに変換する必要があるためである。また、INADDR_ANY は自動的に IPv4 アドレスを決定させるための定数であり、INADDR_*はホストバイトオーダで定義されているため htonl でネットワークバイトオーダに変換する必要がある。

（例題１）の解答例（つづき）

sockets/sind-echoc00.c: AF_INET, SOCK_DGRAM, echo クライアント

```
        :
#include <sys/types.h>  /* sys/socket.h, etc */
#include <sys/socket.h> /* socket, etc */
#include <unistd.h>     /* read, write, close, etc */
#include <netinet/in.h> /* struct sockaddr_in, etc */
#include <netdb.h>      /* gethostbyname, etc */
        :
int main(int argc, char *argv[])
{
  int sfd;
  struct sockaddr_in server;
  socklen_t addr_size;
  in_port_t port = 50000;
  char *host = "localhost", buf[1024];
  struct hostent *he;
        :
  if ((sfd = socket(AF_INET, SOCK_DGRAM, 0)) == -1) { perror("client: socket"); exit(1); }
  memset(&server, 0, sizeof(server));
  server.sin_family = AF_INET;
  server.sin_port = htons(port);
  if ((he = gethostbyname(host)) == NULL) { perror("client: gethostbyname"); exit(1); }
  memcpy(&server.sin_addr, he->h_addr, he->h_length);
  while (fgets(buf, sizeof(buf), stdin)) {
    if (sendto(sfd, buf, sizeof(buf), 0, (struct sockaddr *)&server, sizeof(server)) == -1) { perror("client: sendto"); exit(1); }
    addr_size = sizeof(server);
    if ((recvfrom(sfd, buf, sizeof(buf), 0, (struct sockaddr *)&server, &addr_size)) == -1) { perror("client: recvfrom"); exit(1); }
    printf("%s", buf);
  }
  close(sfd);
  return 0;
}
```

gethostbyname はホスト名もしくは IP アドレスを表す文字列から対応する IP アドレスのリストを返す関数である。リストであるので複数の IP アドレスから選択する余地があるわけだが、ここではレガシーなやり方 he->h_addr(これは he->h_addr_list[0]) で澄ませている。そもそも、新たな状況に対応するには、後に説明する getaddrinfo を用いるべきである。

（例題１）の解答例：ストリーム

sockets/suns-echod.c: AF_UNIX, SOCK_STREAM, echo **サーバ**

```
        :
#include <sys/types.h>  /* sys/socket.h, etc */
#include <sys/socket.h> /* socket, etc */
#include <unistd.h>     /* read, write, close, etc */
#include <sys/un.h>     /* struct sockaddr_un, etc */
        :
int main(int argc, char *argv[])
{
  int sfd, cfd;
  struct sockaddr_un server, client;
  socklen_t addr_size;
  char *sun_path = "/tmp/suns-echo.socket", buf[1024];
        :
  if ((sfd = socket(AF_UNIX, SOCK_STREAM, 0)) == -1) { perror("server: socket"); exit(1); }
  memset(&server, 0, sizeof(server));
  server.sun_family = AF_UNIX;
  strncpy(server.sun_path, sun_path, sizeof(server.sun_path)-1);
  unlink(server.sun_path);
  if (bind(sfd, (struct sockaddr *)&server, sizeof(server)) == -1) { perror("server: bind"); exit(1); }
  if (listen(sfd, 4) == -1) { perror("server: listen"); exit(1); }
  memset(&client, 0, sizeof(client));
  addr_size = sizeof(client);
  if ((cfd = accept(sfd, (struct sockaddr *)&client, &addr_size)) == -1) { perror("server: accept"); exit(1); }
  close(sfd);
  while (!0) {
    if (readln(cfd, buf, sizeof(buf)) == -1) { perror("server: read"); exit(1); }
    if (write(cfd, buf, strlen(buf)) == -1) { perror("server: write"); exit(1); }
  }
  close(cfd);
  unlink(sun_path);
  return 0;
}
```

accept で単一の接続済みソケット cfd を取得した後には、この例ではリスニングソケット sfd は不要となるので、すぐさま close(sfd) している。ちなみに readln は、read で改行等まで一文字ずつ読み込む、ここだけの例示向けの非効率な関数である。

（例題１）の解答例（つづき）

sockets/suns-echoc.c: AF_UNIX, SOCK_STREAM, echo **クライアント**

```
        :
#include <sys/types.h>  /* sys/socket.h, etc */
#include <sys/socket.h> /* socket, etc */
#include <unistd.h>     /* read, write, close, etc */
#include <sys/un.h>     /* struct sockaddr_un, etc */
        :
int main(int argc, char *argv[])
{
  int sfd;
  struct sockaddr_un server;
  char *sun_path = "/tmp/suns-echo.socket", buf[1024];
        :
  if ((sfd = socket(AF_UNIX, SOCK_STREAM, 0)) == -1) { perror("client: socket"); exit(1); }
  memset(&server, 0, sizeof(server));
  server.sun_family = AF_UNIX;
  strncpy(server.sun_path, sun_path, sizeof(server.sun_path)-1);
  if (connect(sfd, (struct sockaddr *)&server, sizeof(server)) == -1) { perror("client: connect"); exit(1); }
  while (fgets(buf, sizeof(buf), stdin)) {
    if (write(sfd, buf, strlen(buf)) == -1) { perror("client; write"); exit(1); }
    if (readln(sfd, buf, sizeof(buf)) == -1) { perror("client: read"); exit(1); }
    printf("%s", buf);
  }
  close(sfd);
  return 0;
}
```

（例題１）の解答例（つづき）

sockets/sins-echod00.c: AF_INET, SOCK_STREAM, echo **サーバ**

```
        :
#include <sys/types.h>  /* sys/socket.h, etc */
#include <sys/socket.h> /* socket, etc */
#include <unistd.h>     /* read, write, close, etc */
#include <netinet/in.h> /* struct sockaddr_in, etc */
        :
int main(int argc, char *argv[])
{
  int sfd, cfd, so;
  struct sockaddr_in server, client;
  socklen_t addr_size;
  in_port_t port = 50000;
  char buf[1024];
        :
  if ((sfd = socket(AF_INET, SOCK_STREAM, 0)) == -1) { perror("server: socket"); exit(1); }
  if (setsockopt(sfd, SOL_SOCKET, SO_REUSEADDR, (void *)&so, sizeof(so))) { perror("server: setsockopt"); exit(1); }
  memset(&server, 0, sizeof(server));
  server.sin_family = AF_INET;
  server.sin_port = htons(port);
  server.sin_addr.s_addr = htonl(INADDR_ANY);
  if (bind(sfd, (struct sockaddr *)&server, sizeof(server)) == -1) { perror("server: bind"); exit(1); }
  if (listen(sfd, 4) == -1) { perror("server: listen"); exit(1); }
  memset(&client, 0, sizeof(client));
  addr_size = sizeof(client);
  if ((cfd = accept(sfd, (struct sockaddr *)&client, &addr_size)) == -1) { perror("server: accept"); exit(1); }
  close(sfd);
  while (!0) {
    if (readln(cfd, buf, sizeof(buf)) == -1) { perror("server: read"); exit(1); }
    if (write(cfd, buf, strlen(buf)) == -1) { perror("server: write"); exit(1); }
  }
  close(cfd);
  return 0;
}
```

`setsockopt(sfd,SOL_SOCKET,SO_REUSEADDR,)` **でソケットに対してそのようなオプションを設定しているのは、ごく簡単には、つづく** `bind` **に失敗しないためである。**

（例題１）の解答例（つづき）

sockets/sins-echoc00.c: AF_INET, SOCK_STREAM, echo **クライアント**

```
        :
#include <sys/types.h>  /* sys/socket.h, etc */
#include <sys/socket.h> /* socket, etc */
#include <unistd.h>     /* read, write, close, etc */
#include <netinet/in.h> /* struct sockaddr_in, etc */
#include <netdb.h>      /* gethostbyname, etc */
        :
int main(int argc, char *argv[])
{
  int sfd;
  struct sockaddr_in server;
  in_port_t port = 50000;
  char *host = "localhost", buf[1024];
  struct hostent *he;
        :
  if ((sfd = socket(AF_INET, SOCK_STREAM, 0)) == -1) { perror("client: socket"); exit(1); }
  memset(&server, 0, sizeof(server));
  server.sin_family = AF_INET;
  server.sin_port = htons(port);
  if ((he = gethostbyname(host)) == NULL) { perror("client: gethostbyname"); exit(1); }
  memcpy(&server.sin_addr, he->h_addr, he->h_length);
  if (connect(sfd, (struct sockaddr *)&server, sizeof(server)) == -1) { perror("client: connect"); exit(1); }
  while (fgets(buf, sizeof(buf), stdin)) {
    if (write(sfd, buf, strlen(buf)) == -1) { perror("client; write"); exit(1); }
    if (readln(sfd, buf, sizeof(buf)) == -1) { perror("client: read"); exit(1); }
    printf("%s", buf);
  }
  close(sfd);
  return 0;
}
```

（例題１）の解答例 DGRAM クライアント別解

sockets/sund-echoc-connect.c: AF_UNIX, SOCK_DGRAM, echo **クライアント**
sockets/sind-echoc-connect.c: AF_INET, SOCK_DGRAM, echo **クライアント**

```
$ diff -u sund-echoc.c sund-echoc-connect.c
$ diff -u sind-echoc.c sind-echoc-connect.c
      :
   memset(&server, 0, sizeof(server));
   server.sun_family = AF_UNIX;
   strncpy(server.sun_path, sun_path, sizeof(server.sun_path)-1);
+  if (connect(sfd, (struct sockaddr *)&server, sizeof(server)) == -1) { perror("client: connect"); exit(1); }

   while (fgets(buf, sizeof(buf), stdin)) {
-    if (sendto(sfd, buf, sizeof(buf), 0, (struct sockaddr *)&server, sizeof(server)) == -1) { perror("client; sendto"); exit(1); }
+    if (write(sfd, buf, sizeof(buf)) == -1) { perror("client; write"); exit(1); }
-    addr_size = sizeof(server);
-    if (recvfrom(sfd, buf, sizeof(buf), 0, (struct sockaddr *)&server, &addr_size) == -1) { perror("client: recvfrom"); exit(1); }
+    if (read(sfd, buf, sizeof(buf)) == -1) { perror("client: read"); exit(1); }
     printf("%s", buf);
```

このように SOCK_DGRAM でも connect を用いてピアを固定することができる。しかし SOCK_STREAM とは異なり３ウェイハンドシェークが行なわれるわけではなく、単にピアのアドレスを保持するだけである。つまり、この機能を表す適切な関数名は setpeername ということになり、同様に bind の機能を表す適切な関数名は setsockname である、という意見もある。

（例題１）の解答例（つづき）IPv4アドレスの表示

sockets/sind-echod.c : AF_INET, SOCK_DGRAM , echo **サーバ**
sockets/sind-echoc*.c: AF_INET, SOCK_DGRAM , echo **クライアント**
sockets/sins-echod.c : AF_INET, SOCK_STREAM, echo **サーバ**
sockets/sins-echoc.c : AF_INET, SOCK_STREAM, echo **クライアント**

```
#include <netdb.h>      /* gethostbyaddr, etc */
#include <arpa/inet.h>  /* inet_ntoa, etc */

void fprint_addrinfo(FILE *fp, const struct sockaddr *sa)
{
  struct in_addr addr = ((struct sockaddr_in *)sa)->sin_addr;
  in_port_t port = ((struct sockaddr_in *)sa)->sin_port;
  struct hostent *he;
  char host[64] = "", name[1024] = "unknown";

#if !defined(HAVE_INET_NTOP)
  strncpy(host, inet_ntoa(addr), sizeof(host)-1); host[sizeof(host)-1] = '\0';
#else
  inet_ntop(AF_INET, &addr, host, sizeof(host));
#endif
  if ((he = gethostbyaddr((char *)&addr, sizeof(struct in_addr), AF_INET))) {
    strncpy(name, he->h_name, sizeof(name)-1); host[sizeof(host)-1] = '\0';
  }
  fprintf(fp, "%s[%s]:%hu\n", name, host, ntohs(port));
}
        :
    fprint_addrinfo(stderr, (struct sockaddr *)&client);
        :
```

このように gethostbyname **に対応する** gethostbyaddr **を用いて、**IP **アドレスからホスト名** he->h_name **を取得している。しかし、新たな状況に対応するには、後に説明する** getnameinfo **を用いるべきである。**

（例題１）の解答例：データグラムIPv6

sockets/sand-echod00.c: AF_INET/AF_INET6, SOCK_DGRAM, echo サーバ

```
        :
#include <sys/types.h>  /* sys/socket.h, etc */
#include <sys/socket.h> /* socket, etc */
#include <unistd.h>     /* read, write, close, etc */
#include <netdb.h>      /* getaddrinfo, etc */
        :
int main(int argc, char *argv[])
{
  int sfd, gai_errno;
  struct addrinfo hints, *res, *r;
  struct sockaddr_storage client;
  socklen_t addr_size;
  char *port = "50000", buf[1024];
        :
  memset(&hints, 0, sizeof(struct addrinfo));
  hints.ai_family = AF_UNSPEC;  /* any address family */
  hints.ai_socktype = SOCK_DGRAM;
  hints.ai_flags |= AI_PASSIVE; /* for bind(2) */
  if ((gai_errno = getaddrinfo(NULL, port, &hints, &res)) != 0) { fprintf(stderr, "server: getaddrinfo: %s\n", gai_strerror(gai_errno)); exit(1); }
  for (r=res; r!=NULL; r=r->ai_next) {
    if ((sfd = socket(r->ai_family, r->ai_socktype, r->ai_protocol)) == -1)
      continue;
    if (bind(sfd, r->ai_addr, r->ai_addrlen) != -1)
      break;
    close(sfd);
  }
  if (!r) { fprintf(stderr, "server: socket, bind\n"); exit(1); }
  freeaddrinfo(res);
  while (!0) {
        : /* 他は sind-echod と同じ */
```

このように getaddrinfo を用いると、hints に指定したアドレスファミリ、タイプ、フラグ (AI_PASSIVE か否か) で適切なアドレスのリストを取得することが出来る。よって、そのリストの中から socket(必要に応じて bind や connect) が真っ先に成功するアドレスを採用すればよいことになる。IPv6 に対応したシステムでは、ここではアドレスファミリに AF_UNSPEC を指定しているので、(環境設定によるが) 優先的な INET6 が選ばれるはずである。

（例題１）の解答例（つづき）

sockets/sand-echoc00.c: AF_INET/AF_INET6, SOCK_DGRAM, echo **クライアント**

```
        :
#include <sys/types.h>  /* sys/socket.h, etc */
#include <sys/socket.h> /* socket, etc */
#include <unistd.h>     /* read, write, close, etc */
#include <netdb.h>      /* getaddrinfo, etc */
        :
int main(int argc, char *argv[])
{
  int sfd, gai_errno;
  struct addrinfo hints, *res, *r;
  struct sockaddr_storage server;
  socklen_t server_addr_size, addr_size;
  char *port = "50000", *host = "localhost", buf[1024];
        :
  memset(&hints, 0, sizeof(struct addrinfo));
  hints.ai_family = AF_UNSPEC;  /* any address family */
  hints.ai_socktype = SOCK_DGRAM;
  if ((gai_errno = getaddrinfo(host, port, &hints, &res)) != 0) { fprintf(stderr, "client: getaddrinfo: %s\n", gai_strerror(gai_errno)); exit(1); }
  for (r=res; r!=NULL; r=r->ai_next) {
    if ((sfd = socket(r->ai_family, r->ai_socktype, r->ai_protocol)) != -1) {
      memcpy(&server, r->ai_addr, r->ai_addrlen);
      server_addr_size = r->ai_addrlen;
      break;
    }
    close(sfd);
  }
  if (!r) { fprintf(stderr, "client: socket\n"); exit(1); }
  freeaddrinfo(res);
  while (fgets(buf, sizeof(buf), stdin)) {
    if (sendto(sfd, buf, sizeof(buf), 0, (struct sockaddr *)&server, server_addr_size) == -1) { perror("client; sendto"); exit(1); }
        : /* 他は sind-echoc と同じ */
```

getaddrinfo において、hints に指定するフラグに AI_PASSIVE を指定していないので、アクティブオープン、つまり、通常のクライアントのソケットのアドレスを想定したアドレスのリストを取得することになる。さらにサーバの例とは異なり、getaddrinfo の第一引数に NULL ではなくサーバのホスト名をしていることにも注目しよう。

（例題１）の解答例：ストリームIPv6

sockets/sans-echod00.c: AF_INET/AF_INET6, SOCK_STREAM, echo サーバ

```
        :
#include <sys/types.h>  /* sys/socket.h, etc */
#include <sys/socket.h> /* socket, etc */
#include <unistd.h>     /* read, write, close, etc */
#include <netdb.h>      /* getaddrinfo, etc */
        :
int main(int argc, char *argv[])
{
  int sfd, cfd, so, gai_errno;
  struct addrinfo hints, *res, *r;
  struct sockaddr_storage client;
  socklen_t addr_size;
  char *port = "50000", buf[1024];
        :
  memset(&hints, 0, sizeof(struct addrinfo));
  hints.ai_family = AF_UNSPEC;  /* any address family */
  hints.ai_socktype = SOCK_STREAM;
  hints.ai_flags |= AI_PASSIVE; /* for bind(2) */
  if ((gai_errno = getaddrinfo(NULL, port, &hints, &res)) != 0) { fprintf(stderr, "server: getaddrinfo: %s\n", gai_strerror(gai_errno)); exit(1); }
  for (r=res; r!=NULL; r=r->ai_next) {
    if ((sfd = socket(r->ai_family, r->ai_socktype, r->ai_protocol)) == -1)
      continue;
    if (bind(sfd, r->ai_addr, r->ai_addrlen) != -1)
      break;
    close(sfd);
  }
  if (!r) { fprintf(stderr, "server: socket, bind\n"); exit(1); }
  freeaddrinfo(res);
  if (setsockopt(sfd, SOL_SOCKET, SO_REUSEADDR, (void *)&so, sizeof(so))) { perror("server: setsockopt"); exit(1); }
  if (listen(sfd, 4) == -1) { perror("server: listen"); exit(1); }
        : /* 他は sins-echod と同じ */
```

freeaddrinfo(res) までは sand-echod00.c とほぼ同じで、本質的な違いは SOCK_DGRAM か SOCK_STREAM だけである。

（例題１）の解答例（つづき）

sockets/sans-echoc00.c: AF_INET/AF_INET6, SOCK_STREAM, echo **クライアント**

```
        :
#include <sys/types.h>  /* sys/socket.h, etc */
#include <sys/socket.h> /* socket, etc */
#include <unistd.h>     /* read, write, close, etc */
#include <netdb.h>      /* gethostbyname, etc */
        :
int main(int argc, char *argv[])
{
  int sfd, gai_errno;
  struct addrinfo hints, *res, *r;
  char *port = "50000", *host = "localhost", buf[1024];
        :
  memset(&hints, 0, sizeof(struct addrinfo));
  hints.ai_family = AF_UNSPEC;  /* any address family */
  hints.ai_socktype = SOCK_STREAM;
  if ((gai_errno = getaddrinfo(host, port, &hints, &res)) != 0) { fprintf(stderr, "client: getaddrinfo: %s\n", gai_strerror(gai_errno)); exit(1); }
  for (r=res; r!=NULL; r=r->ai_next) {
    if ((sfd = socket(r->ai_family, r->ai_socktype, r->ai_protocol)) == -1)
      continue;
    if (connect(sfd, r->ai_addr, r->ai_addrlen) != -1)
      break;
    close(sfd);
  }
  if (!r) { fprintf(stderr, "client: socket, connect\n"); exit(1); }
  freeaddrinfo(res);
  while (fgets(buf, sizeof(buf), stdin)) {
        : /* 他は sins-echoc と同じ */
```

freeaddrinfo(res) **までは** sand-echoc00.c **と** connect **の箇所を除いてほぼ同じで、本質的な違いは** SOCK_DGRAM **か** SOCK_STREAM **だけである。**

（例題１）の解答例 DGRAM クライアント別解

sockets/sand-echoc-connect.c: AF_INET/AF_INET6, SOCK_DGRAM, echo **クライアント**

```
$ diff -u sand-echoc00.c sand-echoc-connect00.c
        :
     exit(1);
   }
   for (r=res; r!=NULL; r=r->ai_next) {
-    if ((sfd = socket(r->ai_family, r->ai_socktype, r->ai_protocol)) != -1) {
-      memcpy(&server, r->ai_addr, r->ai_addrlen);
-      server_addr_size = r->ai_addrlen;
-      break;
-    }
+    if ((sfd = socket(r->ai_family, r->ai_socktype, r->ai_protocol)) == -1)
+      continue;
+    if (connect(sfd, r->ai_addr, r->ai_addrlen) != -1)
+      break;
     close(sfd);
   }
        :
   while (fgets(buf, sizeof(buf), stdin)) {
-    if (sendto(sfd, buf, sizeof(buf), 0, (struct sockaddr *)&server, server_addr_size) == -1) { perror("client: sendto"); exit(1); }
+    if (write(sfd, buf, sizeof(buf)) == -1) { perror("client; write"); exit(1); }
-    addr_size = sizeof(server);
-    if (recvfrom(sfd, buf, sizeof(buf), 0, (struct sockaddr *)&server, &addr_size) == -1) { perror("client: recvfrom"); exit(1); }
+    if (read(sfd, buf, sizeof(buf)) == -1) { perror("client: read"); exit(1); }
     printf("%s", buf);
   }
        :
```

先の SOCK_DGRAM **における** connect **の** AF_INET/AF_INET6 **版であり、**socket **に加えて** connect **が成功するまでアドレスファミリの選択を行なっている。**

（例題１）の解答例（つづき）IPv6 アドレスの表示

sockets/sand-echod.c : AF_INET/AF_INET6, SOCK_DGRAM , echo **サーバ**
sockets/sand-echoc*.c: AF_INET/AF_INET6, SOCK_DGRAM , echo **クライアント**
sockets/sans-echoc.c : AF_INET/AF_INET6, SOCK_STREAM, echo **サーバ**
sockets/sans-echod.c : AF_INET/AF_INET6, SOCK_STREAM, echo **クライアント**

```
void fprint_addrinfo(FILE *fp, const struct sockaddr *sa, socklen_t salen)
{
  int gai_errno;
  char host[NI_MAXHOST], name[NI_MAXHOST] = "unknown", serv[NI_MAXSERV];

  if ((gai_errno=getnameinfo(sa, salen, host, sizeof(host), serv, sizeof(serv),
                             NI_NUMERICHOST|NI_NUMERICSERV)) != 0)
    fprintf(fp, "getnameinfo: %s\n", gai_strerror(gai_errno));
  else {
    if ((gai_errno=getnameinfo(sa, salen, name, sizeof(name), NULL,
                               0, NI_NAMEREQD)) != 0)
      ;
    fprintf(fp, "%s[%s]:%s\n", name, host, serv);
  }
}
        :
    fprint_addrinfo(stderr, (struct sockaddr *)&client, addr_size);
        :
```

このように `getaddrinfo` **に対応する** `getnameinfo` **を用いて、まず、アドレス** `host` **とポート** `serv`**(この名はサービスに由来している)を数値表現(**`NI_NUMERICHOST|NI_NUMERICSERV`**)の文字列で取得し、つづいて、ホスト名** `name` **を取得している。その際に** `NI_NAMEREQD` **を指定して、ホスト名が取得できないときにエラーを返すようにしている。**

ソケットコードの要（1／3）Unix ドメイン

	server	*client*	*client*

UNIX DGRAM	/* *Socket UNIX DGRAM server: sund-echod* */
	struct sockaddr_un server, client;
	sfd=socket(AF_UNIX, SOCK_DGRAM,);
	strlcpy(server.sun_path,sun_path,);
	unlink(server.sun_path);
	bind(sfd, &server,);
	
	recvfrom(sfd,..,&client,);
	sendto(sfd,..,&client,);
		/* *Socket UNIX DGRAM client: sund-echoc* */
		struct sockaddr_un server, client;
		sfd=socket(AF_UNIX, SOCK_DGRAM,);
		client.sun_family = AF_UNIX;
		strlcpy(client.sun_path, tempnam("/tmp",),);
		bind(sfd, &client,);
		server.sun_family = AF_UNIX;
		strlcpy(server.sun_path, sun_path,);
		
		sendto(sfd,..,&server,);
		recvfrom(sfd,..,&server,);
			/* *Socket UNIX DGRAM client: sund-echoc* */
			struct sockaddr_un server, client;
			sfd=socket(AF_UNIX, SOCK_DGRAM,);
			client.sun_family = AF_UNIX;
			strlcpy(client.sun_path, tempnam("/tmp",),);
			bind(sfd, &client,);
			server.sun_family = AF_UNIX;
			strlcpy(server.sun_path, sun_path,);
			connect(sfd, &server,);
			
			write(sfd,);
			read(sfd,);

UNIX STREAM	/* *Socket UNIX STREAM server: suns-echod* */
	struct sockaddr_un server, client;
	sfd=socket(AF_UNIX, SOCK_STREAM,);
	strlcpy(server.sun_path,sun_path,);
	unlink(server.sun_path);
	bind(sfd, &server,); listen(sfd,);
	cfd=accept(sfd,&client,);
	
	read(cfd,);
	write(cfd,);
		/* *Socket UNIX STREAM client: suns-echoc* */
		struct sockaddr_un server;
		sfd=socket(AF_UNIX, SOCK_STREAM,);
		strlcpy(server.sun_path, sun_path,);
		
		connect(sfd, &server,);
		
		write(sfd,);
		read(sfd,);	

ソケットコードの要（２／３）INET

	server	*client*	*client*

INET DGRAM	/* *Socket INET DGRAM server: sind-echod* */
	struct sockaddr_in server, client;
	sfd=**socket(AF_INET, SOCK_DGRAM,);**
	server.sin_family = AF_INET;
	server.sin_port = htons(port);
	server.sin_addr.s_addr = htonl(INADDR_ANY);
	bind(sfd, &server,);
	
	recvfrom(sfd,..,&client,);
	sendto(sfd,..,&client,);
		/* *Socket INET DGRAM client: sind-echoc* */
		struct sockaddr_in server;
		sfd=**socket(AF_INET, SOCK_DGRAM,);**
		server.sin_family = AF_INET;
		server.sin_port = htons(port);
		memcpy(&server.sin_addr, he->h_addr,);
		
		sendto(sfd,..,&server,);
		recvfrom(sfd,..,&server,);
			/* *Socket INET DGRAM client: sind-echoc-connect* */
			struct sockaddr_in server;
			sfd=**socket(AF_INET, SOCK_DGRAM,);**
			server.sin_family = AF_INET;
			server.sin_port = htons(port);
			memcpy(&server.sin_addr, he->h_addr,);
			connect(sfd, &server,);
			
			write(sfd,);
			read(sfd,);

INET STREAM	/* *Socket INET STREAM server: sins-echod* */
	struct sockaddr_in server, client;
	sfd=**socket(AF_INET, SOCK_STREAM,);**
	server.sin_family = AF_INET;
	server.sin_port = htons(port);
	server.sin_addr.s_addr = htonl(INADDR_ANY);
	bind(sfd, &server,); listen(sfd,);
	cfd=accept(sfd,&client,);
	
	read(cfd,);
	write(cfd,);
		/* *Socket INET STREAM client: sins-echoc* */
		struct sockaddr_in server;
		sfd=**socket(AF_INET, SOCK_STREAM,);**
		server.sin_family = AF_INET;
		server.sin_port = htons(port);
		memcpy(&server.sin_addr, he->h_addr,);
		connect(sfd, &server,);
		
		write(sfd,);
		read(sfd,);	

ソケットコードの要（１／３）INET/INET6

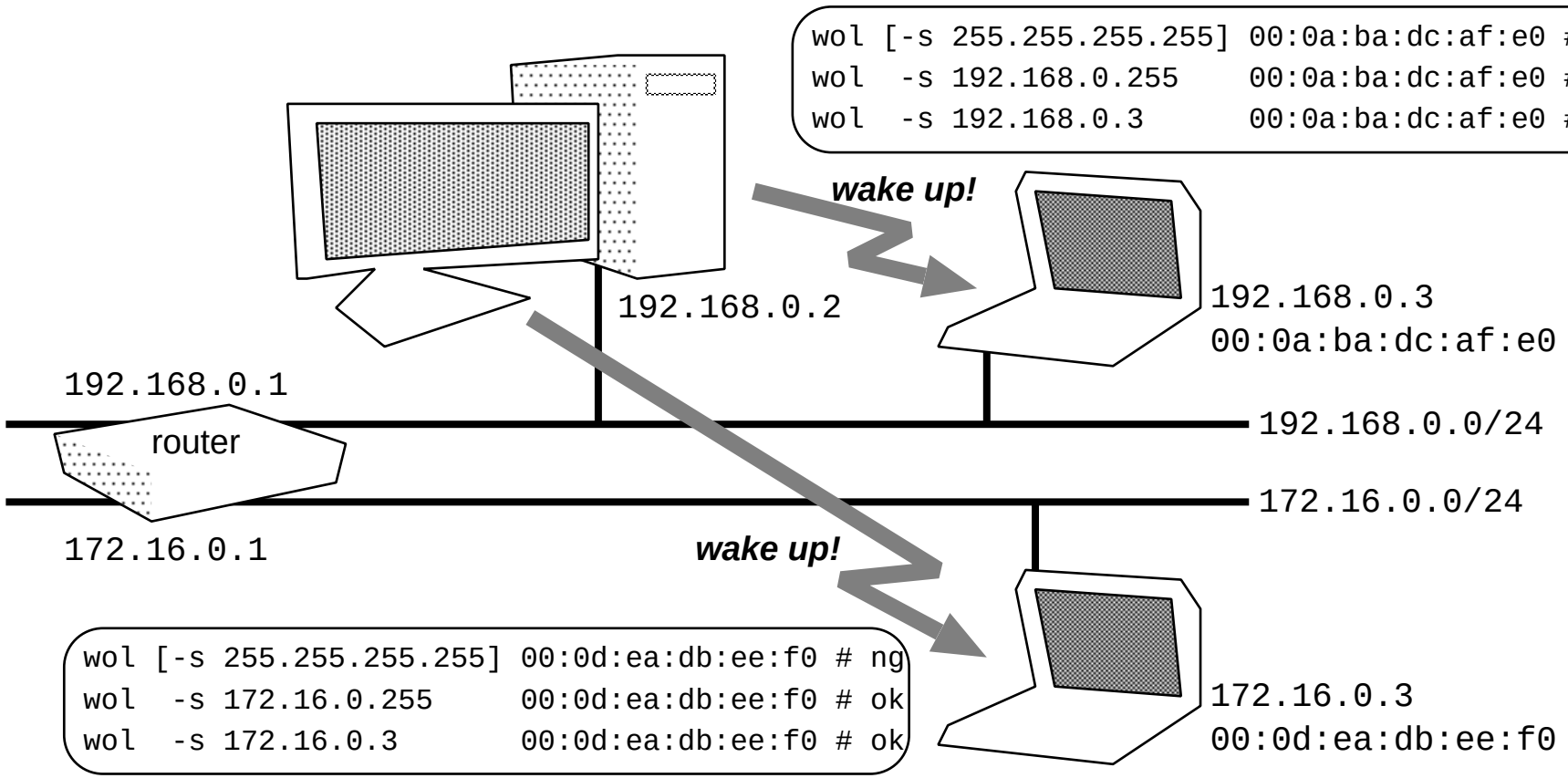
	server	client	client

INET/INET6 DGRAM	/* Socket INET DGRAM server: sand-echod */
	hints.ai_family = AF_UNSPEC;
	hints.ai_socktype = SOCK_DGRAM;
	hints.ai_flags |= AI_PASSIVE;
	getaddrinfo(NULL, port, &hints, &res);
	for (r=res; r!=NULL; r=r->ai_next) {
	if ((sfd=socket(r->ai_family, r->ai_socktype,))!=-1) {
	bind(sfd, r->ai_addr, r->ai_addrlen);
	break;
	}
	close(sfd);
	}
	recvfrom(sfd,..,&client,);
	sendto(sfd,..,&client,);
		/* Socket INET DGRAM client: sand-echoc */
		hints.ai_family = AF_UNSPEC;
		hints.ai_socktype = SOCK_DGRAM;
		getaddrinfo(host, port, &hints, &res);
		for (r=res; r!=NULL; r=r->ai_next) {
		if ((sfd=socket(r->ai_family, r->ai_socktype,))!=-1) {
		memcpy(&server, r->ai_addr, r->ai_addrlen);
		server_addr_size = r->ai_addrlen;
		break;
		}
		close(sfd);
		}
		sendto(sfd,..,&server,);
		recvfrom(sfd,..,&server,);
			/* Socket INET DGRAM client: sand-echoc-connect */
			hints.ai_family = AF_UNSPEC;
			hints.ai_socktype = SOCK_DGRAM;
			getaddrinfo(host, port, &hints, &res);
			for (r=res; r!=NULL; r=r->ai_next) {
			if ((sfd=socket(r->ai_family, r->ai_socktype,))!=-1) {
			connect(sfd, r->ai_addr, r->ai_addrlen);
			break;
			}
			close(sfd);
			}
			
			write(sfd,);
			read(sfd,);

INET/INET6 STREAM	/* Socket INET STREAM server: sans-echod */
	hints.ai_family = AF_UNSPEC;
	hints.ai_socktype = SOCK_STREAM;
	hints.ai_flags |= AI_PASSIVE;
	getaddrinfo(NULL, port, &hints, &res);
	for (r=res; r!=NULL; r=r->ai_next) {
	if ((sfd=socket(r->ai_family, r->ai_socktype,))!=-1) {
	bind(sfd, r->ai_addr, r->ai_addrlen);
	break;
	}
	close(sfd);
	} listen(sfd,); cfd=accept(sfd,&client,);
	read(cfd,);
	write(cfd,);
		/* Socket INET STREAM client: sans-echoc */
		hints.ai_family = AF_UNSPEC;
		hints.ai_socktype = SOCK_STREAM;
		getaddrinfo(host, port, &hints, &res);
		for (r=res; r!=NULL; r=r->ai_next) {
		if ((sfd=socket(r->ai_family, r->ai_socktype,))!=-1) {
		connect(sfd, r->ai_addr, r->ai_addrlen);
		break;
		}
		close(sfd);
		}
		
		write(sfd,);
		read(sfd,);	

（例題２）IPv4のブロードキャスト

休止状態等にあるネットワークに接続されたマシンに「マジックパケット」と呼ばれる特殊な UDP パケットを送り付けると、そのマシンを活動状態にすることができる WOL(Wake On Lan) と呼ばれる電源管理機構がある。マジックパケットは、管理対象となるマシンのネットワークインターフェースの MAC(Media Access Control) アドレスを 48 ビット「00:0a:ba:dc:af:e0」としたとき、「0xff」を６回、MAC アドレスを１６回つなげたデータが主に使われる。そして、このマジックパケットを UDP ポート 9 として、ブロードキャストアドレスまたは管理対象となるマシンの IPv4 アドレスに送り付けることになる。この機能を実現する `wol` コマンドを作成せよ。



（例題２）の解答例

sockets/wol.c: AF_INET/AF_INET6, SOCK_DGRAM, **ブロードキャスト**

```
        :
#include <sys/types.h>  /* sys/socket.h, etc */
#include <sys/socket.h> /* socket, etc */
#include <unistd.h>     /* read, write, close, etc */
#include <netinet/in.h> /* struct sockaddr_in, etc */
#include <arpa/inet.h>  /* inet_ntoa, etc */
#include <netdb.h>      /* getaddrinfo, etc */
#include <net/if.h>             /* netinet/if_ether.h, etc */
#include <netinet/if_ether.h>   /* ether_aton, etc */
        :
int main(int argc, char *argv[])
{
  int sfd, so, gai_errno, i;
  struct addrinfo hints, *res, *r;
  struct sockaddr_storage peer;
  socklen_t peer_addr_size;
  char buf[1024], *host = "255.255.255.255", *port = "9", *maddr = NULL;
  struct ether_addr eaddr, *ea;
  ssize_t s = 0;
        :
  memset(&hints, 0, sizeof(struct addrinfo));
  hints.ai_family = AF_UNSPEC;  /* any address family */
  hints.ai_socktype = SOCK_DGRAM;
  if ((gai_errno = getaddrinfo(host, port, &hints, &res)) != 0) { fprintf(stderr, "wol: getaddrinfo: %s\n", gai_strerror(gai_errno)); exit(1); }
  for (r=res; r!=NULL; r=r->ai_next) {
    if ((sfd = socket(r->ai_family, r->ai_socktype, r->ai_protocol)) != -1) {
      memcpy(&peer, r->ai_addr, r->ai_addrlen);
      peer_addr_size = r->ai_addrlen;
      break;
    }
    close(sfd);
  }
  if (!r) { fprintf(stderr, "wol: socket\n"); exit(1); }
  freeaddrinfo(res);    /* ここまでは、基本的に sand-echoc.c と同じ */
```

（例題２）の解答例（つづき）

sockets/wol.c: AF_INET/AF_INET6, SOCK_DGRAM, ブロードキャスト

```
    if (setsockopt(sfd, SOL_SOCKET, SO_BROADCAST, (void *)&so, sizeof(so))) { // ブロードキャストパケットの許可
      perror("wol: setsockopt");
      exit(1);
    }
    /* make magic packet */
    if ((ea = ether_aton(maddr)))                // MAC アドレス（イーサネットアドレス）文字列から struct ether_addr *の取得
      eaddr = *ea;
    else if (ether_hostton(maddr, &eaddr) == 0)  // 失敗した（ea==NULL）場合/etc/ethers に登録されたホスト名として eaddr へ取得
      ;
    else {
      fprintf(stderr, "wol: illegal ethernet address for %s\n", maddr);
      exit(1);
    }
    memset(&buf[0]+s, 0xff, 6); s+=6;
    for (i=0; i<16; i++) {
      memcpy(&buf[0]+s, eaddr.ether_addr_octet, 6); s+=6;
    }                                            // buf にマジックパケット「0xff × 6 ＋ eaddr.ether_addr_octet × 16」が格納、長さは s
    if (sendto(sfd, buf, s, 0, (struct sockaddr *)&peer, peer_addr_size) != s) {
      perror("wol: sendto");
      exit(1);
    }
    return 0;
  }
```

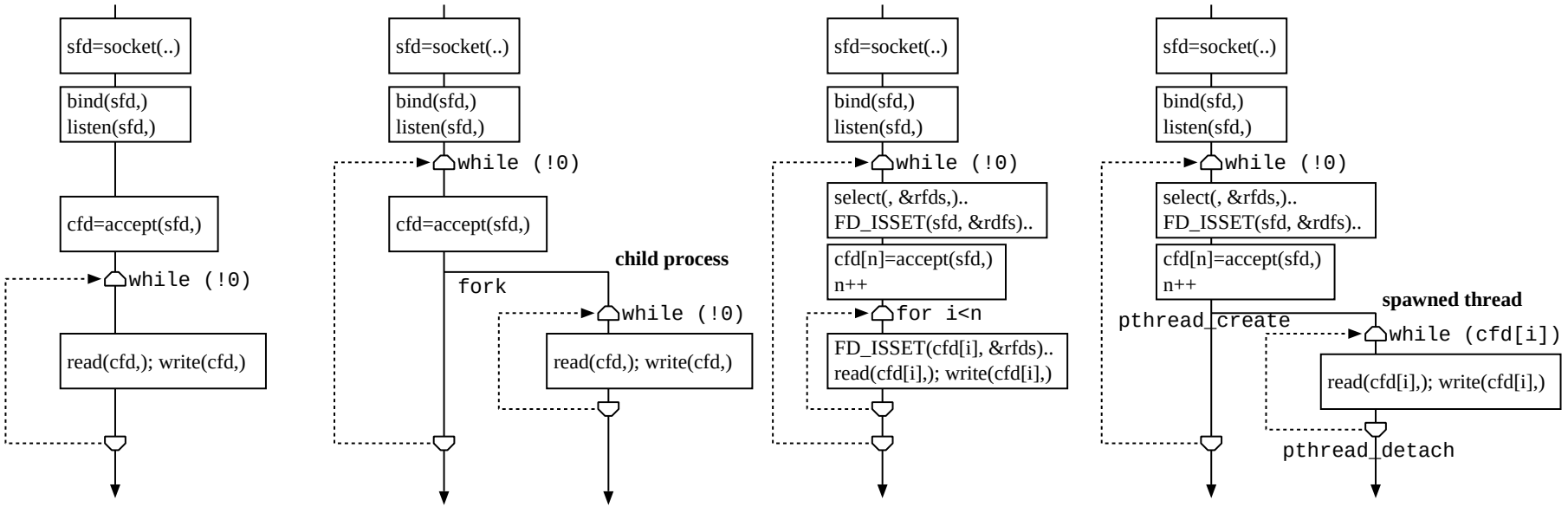
```
$ make CC=gcc LDLIBS='-lsocket -lnsl' wol
$ ./wol
usage:
        wol [-h] [-s peer] [-p port] ethernet-address
$ ./wol                  00:0a:ba:dc:af:e0
$ ./wol -s 192.168.0.255 00:0a:ba:dc:af:e0
$ ./wol -s 192.168.0.3   00:0a:ba:dc:af:e0
```

（例題３）マルチクライアント対応

先の sins-echod、sans-echod などは、単一の接続済みソケットしか accept で取得していないので、複数のクライアントに対応していない。よって、まずは sins-echod、sans-echod を fork を使って複数のクライアントの接続要求を accept するようにせよ (sins-echod-fork、sans-echod-fork)。

次に、まずは fork 等を使わずに、I/O を多重化する select を用いて echo サーバ／クライアントを発展させた chat サーバ／クライアントを作成せよ (sins-chatd、sins-chatc)。

発展として、(fork でもよいが、ここは)pthreads を使った chat サーバを作成せよ。



（例題３）の解答例（sins-echod-fork.c）

```
        : /* 他は sins-echod.c と同じ */
  if (listen(sfd, 4) == -1) { perror("server: listen"); exit(1); }
  while (!0) {
    memset(&client, 0, sizeof(client));
    addr_size = sizeof(client);
    if ((cfd = accept(sfd, (struct sockaddr *)&client, &addr_size)) == -1) { perror("server: accept"); exit(1); }
    else {
      pid_t pid;

      fprint_addrinfo(stderr, (struct sockaddr *)&client);
      if ((pid=fork()) < 0) { perror("server: fork"); exit(1); }
      else if (pid == 0) { /* child process */
        close(sfd);
        while (!0) {
          if (readln(cfd, buf, sizeof(buf)) == -1) { perror("server: read"); exit(1); }
          if (write(cfd, buf, strlen(buf)) == -1) { perror("server: write"); exit(1); }
        }
        close(cfd);
        exit(1);
      }
      close(cfd);
    }
  }
  close(sfd);
  return 0;
}
```

```
$ make CC=gcc LDLIBS='-lsocket -lnsl' sins-echod-fork sins-echoc
$ ./sins-echod-fork
$ ./sins-echoc          # 別の端末で
$ ./sins-echoc          # さらに別の端末で
```

（例題３）の解答例（sins-chatd.c）

```
        :
#include <sys/types.h>  /* sys/socket.h, etc */
#include <sys/socket.h> /* socket, etc */
#include <unistd.h>     /* read, write, close, etc */
#include <netinet/in.h> /* struct sockaddr_in, etc */
#include <strings.h>    /* strcasecmp, etc */
#include <sys/select.h> /* select, etc */
#include <netdb.h>      /* gethostbyname, etc */
#include <arpa/inet.h>  /* inet_ntoa, etc */
        :
int main(int argc, char *argv[])
{
  int a, i, j, n_clients = 0;
  int sfd, cfd[N_CLIENTS+1], mfd, so; // リスニングソケット、接続済みソケット用配列 (+1 は予備)、デスクリプタの最大値、ソケットオプション
  struct sockaddr_in server, client;
  socklen_t addr_size;
  in_port_t port = 50000;
  char buf[1024] = "", str[1024];
  struct timeval tv, timeout = { 0, 10 };
  fd_set rfds; // 後の select で読み込み可能か監視するデスクリプタ集合
        :
  if ((sfd = socket(AF_INET, SOCK_STREAM, 0)) == -1) { perror("server: socket"); exit(1); }
  if (setsockopt(sfd, SOL_SOCKET, SO_REUSEADDR, (void *)&so, sizeof(so))) { perror("server: setsockopt"); exit(1); }
  memset(&server, 0, sizeof(server));
  server.sin_family = AF_INET;
  server.sin_port = htons(port);
  server.sin_addr.s_addr = htonl(INADDR_ANY);
  if (bind(sfd, (struct sockaddr *)&server, sizeof(server)) == -1) { perror("server: bind"); exit(1); }
  if (listen(sfd, 4) == -1) { perror("server: listen"); exit(1); } // ここまでは基本的に sins-echod.c と同じ
  for (i=0; i<sizeof(cfd)/sizeof(*cfd); i++) // 接続済みソケット用配列 cfd[] を、未使用の意味で-1 に初期化
    cfd[i] = -1;
        :
```

（例題３）の解答例（sins-chatd.c つづき）

```
      :
while (strcasecmp(buf, "quit\n") != 0) { // buf[] が大文字小文字区別せず"quit\n"でない限り繰り返す
  FD_ZERO(&rfds);                                    // デスクリプタ集合 rfds を初期化
  FD_SET(sfd, &rfds); mfd = sfd;                     // リスニングソケット sfd をデスクリプタ集合に追加、デスクリプタの最大値 mfd を sfd
  FD_SET(0, &rfds);                                  // 標準入力 0:fileno(stdin) をデスクリプタ集合に追加
  for (i=0; i<n_clients; i++) {
    if (cfd[i] == -1) continue;                      // 接続済みソケット cfd[i] が未使用ならば、何もしない
    FD_SET(cfd[i], &rfds); mfd = max(mfd, cfd[i]);   // 接続済みソケット cfd[i] をデスクリプタ集合に追加、デスクリプタの最大値 mfd の算出
  }
  tv = timeout; // tv は select により書き換えられる可能性があるので timeout は直接 select に指定しないこと！
  if (select(mfd+1, &rfds, NULL, NULL, &tv) == -1) { perror("server: select"); exit(1); } // デスクリプタ集合 rfds が読み込み可能か監視
  if (FD_ISSET(sfd, &rfds)) {                        // リスニングソケット sfd が読み込み可能であるなら、
    memset(&client, 0, sizeof(client));
    addr_size = sizeof(client);                      // とりあえず accept して接続済みソケット cfd[n_client] を取得
    if ((cfd[n_clients] = accept(sfd, (struct sockaddr *)&client, &addr_size)) == -1) { perror("server: accept"); exit(1); }
    if (n_clients < N_CLIENTS)                       // n_client が最大数より少なければ、インクリメント
      n_clients++;
    else {                                           // n_client が最大数ならば、
      for (i=0; i<n_clients; i++)
        if (cfd[i] == -1) break;                     // 現在は未使用の接続済みソケット cfd[i] を探す
      if (i < n_clients)
        cfd[i] = cfd[n_clients];                     // 見つかればそこに接続済みソケット cfd[n_client] を割り当てる
      else {                                         // さもなくば、接続拒否通知、予備の接続済みソケットを close
        snprintf(str, sizeof(str), "server: too busy\nbye\n");
        write(cfd[n_clients], str, strlen(str));
        close(cfd[n_clients]);
      }
      cfd[n_clients] = -1;                           // 予備の接続済みソケットを未使用へ
    }
  }
      :
```

（例題３）の解答例（sins-chatd.c つづき）

```
        :
    buf[0] = '\0';
    for (i=0; i<n_clients; i++) {                    // 接続数 n_clients 分ループ
      if (cfd[i] == -1) continue;                    // 接続済みソケット cfd[i] が未使用ならば、何もしない
      if (FD_ISSET(cfd[i], &rfds)) {                 // 接続済みソケット cfd[i] が読み込み可能であるなら、読み込む
        if (readln(cfd[i], buf, sizeof(buf)) == -1 || strcasecmp(buf, "bye\n") == 0) {
          snprintf(buf, sizeof(buf), "connection closed.\n");   // 読み込んでエラー等なら、接続済みソケット cfd[i] を close、未使用へ
          close(cfd[i]);
          cfd[i] = -1;
        }
        else if (strcasecmp(buf, "quit\n") == 0) break;
        snprintf(str, sizeof(str), "client: %d: %s", i, buf);   // 読み込んでエラー等でなければ、出力用文字列 str を作成
        write(1, str, strlen(str));                  // 標準出力 1:fileno(stdout) へ str を出力
        for (j=0; j<n_clients; j++) {                // このクライアント i 以外のすべてのクライアント j に str を出力
          if (cfd[j] == -1 || j == i) continue;
          write(cfd[j], str, strlen(str));
        }
      }
    }
    if (FD_ISSET(0, &rfds)) {                        // 標準入力 0:fileno(stdin) が読み込み可能であるなら、
      if (readln(0, buf, sizeof(buf)) == -1 || strcasecmp(buf, "quit\n") == 0) break; // 読み込んでエラー等なら、while ループから抜ける
      snprintf(str, sizeof(str), "server: %s", buf);            // 読み込んでエラー等でなければ、出力用文字列 str を作成
      for (i=0; i<n_clients; i++) {                  // すべてのクライアント i に str を出力
        if (cfd[i] == -1) continue;
        write(cfd[i], str, strlen(str));
      }
    }
  }
  for (i=0; i<n_clients; i++) {                      // すべての未使用でない接続済みソケット cfd[i] を close
    if (cfd[i] == -1) continue;
    close(cfd[i]);
    cfd[i] = -1;
  }
  close(sfd);                                        // リスニングソケットを close
  return 0;
}
```

（例題３）の解答例（sins-chatc.c）

```
        :
int main(int argc, char *argv[])
{
  int sfd;
  struct sockaddr_in server;
  in_port_t port = 50000;
  char *host = "localhost", buf[1024] = "";
  struct hostent *he;
  struct timeval tv, timeout = { 0, 10 };
  fd_set rfds; // 後の select で読み込み可能か監視するデスクリプタ集合
        :
  if ((sfd = socket(AF_INET, SOCK_STREAM, 0)) == -1) { perror("client: socket"); exit(1); }
  memset(&server, 0, sizeof(server));
  server.sin_family = AF_INET;
  server.sin_port = htons(port);
  if ((he = gethostbyname(host)) == NULL) { perror("client: gethostbyname"); exit(1); }
  memcpy(&server.sin_addr, he->h_addr, he->h_length); // 次の connect までは基本的に sins-echoc.c と同じ
  if (connect(sfd, (struct sockaddr *)&server, sizeof(server)) == -1) { perror("client: connect"); exit(1); }
  while (strcasecmp(buf, "quit\n") != 0) { // buf[] が大文字小文字区別せず"quit\n"でない限り繰り返す
    FD_ZERO(&rfds);                                    // デスクリプタ集合 rfds を初期化
    FD_SET(sfd, &rfds);                                // ソケット sfd をデスクリプタ集合に追加
    FD_SET(0, &rfds);                                  // 標準入力 0:fileno(stdin) をデスクリプタ集合に追加
    tv = timeout; // tv は select により書き換えられる可能性があるので timeout は直接 select に指定しないこと！
    if (select(sfd+1, &rfds, NULL, NULL, &tv) == -1) { perror("client: select"); } // デスクリプタ集合 rfds が読み込み可能か tv 時間監視
    buf[0] = '\0';
    if (FD_ISSET(sfd, &rfds)) {                        // ソケット sfd が読み込み可能であるなら、
      if (readln(sfd, buf, sizeof(buf)) == -1 || strcasecmp(buf, "bye\n") == 0) break; // 読み込んでエラー等なら、while ループから抜ける
      write(1, buf, strlen(buf));                                  // 読み込んでエラー等でなければ、それを標準出力 1:fileno(stdout) へ出力
    }
    if (FD_ISSET(0, &rfds)) {                          // 標準入力 0:fileno(stdin) が読み込み可能であるなら、
      if (readln(0, buf, sizeof(buf)) == -1 || strcasecmp(buf, "bye\n") == 0) break; // 読み込んでエラー等なら、while ループから抜ける
      write(sfd, buf, strlen(buf));                                // 読み込んでエラー等でなければ、それをサーバへ出力
    }
  }
  close(sfd);
  return 0;
}
```

（例題３）の解答例（sins-chatd-thread.c）

```
        :
#include <pthread.h>    /* pthread_create, etc */
        :
struct thc_arg {
  int i, *cfd, *n_clients;
  char *buf;
};
void *thc_proc(void *arg)
{
  int j;
  struct thc_arg *a = (struct thc_arg *)arg;
  char buf[1024], str[1024];

  while (a->cfd[a->i] != -1) { // 接続済みソケット a->cfd[a->i] が未使用でない限り繰り返す
    if (readln(a->cfd[a->i], buf, sizeof(buf)) == -1 || strcasecmp(buf, "bye\n") == 0) {
      snprintf(buf, sizeof(buf), "connection closed.\n");       // 読み込んでエラー等なら、接続済みソケット a->cfd[a->i] を close、未使用へ
      close(a->cfd[a->i]);
      a->cfd[a->i] = -1;
    }
    else if (strcasecmp(buf, "quit\n") == 0) {
      snprintf(a->buf, sizeof(buf), "%s", buf);
      break;
    }
    snprintf(str, sizeof(str), "client: %d: %s", a->i, buf);    // 読み込んでエラー等でなければ、出力用文字列 str を作成
    write(1, str, strlen(str));                          // 標準出力 1:fileno(stdout) へ str を出力
    for (j=0; j<*a->n_clients; j++) {                    // このクライアント a->i 以外のすべてのクライアント j に str を出力
      if (a->cfd[j] == -1 || j == a->i) continue;
      write(a->cfd[j], str, strlen(str));
    }
  }
  free(a);
  pthread_detach(pthread_self());                        // これ自身のスレッドをデタッチ
  return NULL;
}
        :
```

（例題３）の解答例（sins-chatd-thread.c つづき）

```
        :
int main(int argc, char *argv[])
{
  int i, n_clients = 0, sfd, cfd[N_CLIENTS+1], so;
  struct sockaddr_in server, client;
  socklen_t addr_size;
  in_port_t port = 50000;
  char buf[1024] = "", str[1024];
  struct timeval tv, timeout = { 0, 10 };
  fd_set rfds;
  pthread_t thc[N_CLIENTS];
        :
  if ((sfd = socket(AF_INET, SOCK_STREAM, 0)) == -1) { perror("server: socket"); exit(1); }
  if (setsockopt(sfd, SOL_SOCKET, SO_REUSEADDR, (void *)&so, sizeof(so))) { perror("server: setsockopt"); exit(1); }
  memset(&server, 0, sizeof(server));
  server.sin_family = AF_INET;
  server.sin_port = htons(port);
  server.sin_addr.s_addr = htonl(INADDR_ANY);
  if (bind(sfd, (struct sockaddr *)&server, sizeof(server)) == -1) { perror("server: bind"); exit(1); }
  if (listen(sfd, 4) == -1) { perror("server: listen"); exit(1); }
  for (i=0; i<sizeof(cfd)/sizeof(*cfd); i++)
    cfd[i] = -1;                                          // ここまでは sins-chatd.c と基本的に同じ
  while (strcasecmp(buf, "quit\n") != 0) {
    FD_ZERO(&rfds);
    FD_SET(sfd, &rfds);
    FD_SET(0, &rfds);
    for (i=0; i<n_clients; i++) {
      if (cfd[i] == -1) continue;
      FD_SET(cfd[i], &rfds);
    }
    tv = timeout;
    if (select(sfd+1, &rfds, NULL, NULL, &tv) == -1) { perror("server: select"); exit(1); }
    if (FD_ISSET(sfd, &rfds)) {
      memset(&client, 0, sizeof(client));
      addr_size = sizeof(client);
      if ((cfd[n_clients] = accept(sfd, (struct sockaddr *)&client, &addr_size)) == -1) { perror("server: accept"); exit(1); }
        :                                                 // ここまでは sins-chatd.c よりむしろ単純、なぜなら cfd[i] を監視しないため
```

（例題３）の解答例（sins-chatd-thread.c つづき）

```
        :
      if (n_clients < N_CLIENTS)                    // n_client が最大数より少なければ、それを i に保持しつつ、インクリメント
        i = n_clients++;
      else {
        for (i=0; i<n_clients; i++)
          if (cfd[i] == -1) break;
        if (i < n_clients)
          cfd[i] = cfd[n_clients];
        else {
          snprintf(str, sizeof(str), "server: too busy\nbye\n");
          write(cfd[n_clients], str, strlen(str));
          close(cfd[n_clients]);
        }
        cfd[n_clients] = -1;
      }
      if (cfd[i] != -1) {                           // つまり、新規参加ならば
        struct thc_arg arg = { i, cfd, &n_clients, buf },
          *a = malloc(sizeof(struct thc_arg));

        memcpy(a, &arg, sizeof(struct thc_arg));
        pthread_create(&thc[i], NULL, thc_proc, a);     // スレッド手続き thc_proc(a) でスレッドの生成
      }
    }
        : /* 他は sins-chatd と同じ */
```

```
$ make CC=gcc LDLIBS='-lsocket -lnsl' sins-chatd sins-chatd-thread sins-chatc
$ ./sins-chatd          # もしくは ./sins-chatd-thread
$ ./sins-chatc          # 別の端末で
$ ./sins-chatc          # さらに別の端末で
```

（例題３）の解答例 STREAM 別解：高水準入出力

```
$ diff -u sins-chatd.c sins-fchatd.c    # の主要箇所
        :
   fd_set rfds;
+  FILE *ifp[N_CLIENTS+1], *ofp[N_CLIENTS+1];
        :
       if (n_clients < N_CLIENTS)                      // n_client が最大数より少なければ、それを i に保持しつつ、インクリメント
-        n_clients++;
+        i = n_clients++;
       else {
        :
+    if (cfd[i] != -1) {                                // つまり、新規参加ならば
+      ifp[i] = fdopen(cfd[i], "r");                           // デスクリプタ cfd[i] を読み込みモードでストリーム ifp[i] に関連付ける
+      ofp[i] = fdopen(cfd[i], "w");                           // デスクリプタ cfd[i] を書き込みモードでストリーム ofp[i] に関連付ける
+      setvbuf(ofp[i], NULL, _IOLBF, 0);                      // ストリーム ofp[i] を行バッファリングに設定する
+    }
        :
       if (FD_ISSET(cfd[i], &rfds)) {
-        if (readln(cfd[i], buf, sizeof(buf)) == -1 || strcasecmp(buf, "bye\n") == 0) {
+        if (!fgets(buf, sizeof(buf), ifp[i]) || strcasecmp(buf, "bye\n") == 0) {
           snprintf(buf, sizeof(buf), "connection closed.\n");
           close(cfd[i]);
           cfd[i] = -1;
        :
         for (j=0; j<n_clients; j++) {
           if (cfd[j] == -1 || j == i) continue;
-          write(cfd[j], str, strlen(str));
+          fprintf(ofp[j], "%s", str);
         }
        :
       for (i=0; i<n_clients; i++) {
         if (cfd[i] == -1) continue;
-        write(cfd[i], str, strlen(str));
+        fprintf(ofp[i], "%s", str);
       }
        :
```

（例題３）の解答例 STREAM 別解：高水準入出力

```
$ diff -u sins-chatc.c sins-fchatc.c    # の主要箇所
       :
   fd_set rfds;
+  FILE *ifp, *ofp;
       :
+  ifp = fdopen(sfd, "r");                          // デスクリプタ sfd を読み込みモードでストリーム ifp に関連付ける
+  ofp = fdopen(dup(sfd), "w");                     // デスクリプタ sfd を書き込みモードでストリーム ofp に関連付ける
+  setvbuf(ofp, NULL, _IOLBF, 0);                   // ストリーム ofp を行バッファリングに設定する
   while (strcasecmp(buf, "quit\n") != 0) {
       :
     buf[0] = '\0';
     if (FD_ISSET(sfd, &rfds)) {
-      if (readln(sfd, buf, sizeof(buf)) == -1 || strcasecmp(buf, "byte\n") == 0)
+      if (!fgets(buf, sizeof(buf), ifp) || strcasecmp(buf, "byte\n") == 0)
         break;
-      write(1, buf, strlen(buf));
+      fprintf(stdout, "%s", buf);
     }
     if (FD_ISSET(0, &rfds)) {
-      if (readln(0, buf, sizeof(buf)) == -1 || strcasecmp(buf, "byte\n") == 0)
+      if (!fgets(buf, sizeof(buf), stdin) || strcasecmp(buf, "byte\n") == 0)
         break;
-      write(sfd, buf, strlen(buf));
+      fprintf(ofp, "%s", buf);
     }
   }
-  close(sfd);  // 上記の dup(sfd) は必ずしも必要ないが、
+  fclose(ifp);
+  fclose(ofp); // dup(sfd) なしだと、このように実体が同じストリームを再度 fclose するとエラーとなる
   return 0;
 }
```

このように fdopen を用いることで、read/write の代わりに fgets/fprintf 等の高水準入出力が利用できるようになる。するとバッファリング機構により効率的な入出力の恩恵に預かることが期待できる。ここでは setvbuf で行バッファリングを指定している。

（例題４）ソケットとグラフィックス

これまでのソケット API の基礎に加えて、例えば X11 グラフィックスを使ったネットワーク対応グラフィックス応用例に関するプログラムを作成せよ。

解答例 `x11-sockets/xnetminesweeper/*.{hc}`：

```
$ make CC=gcc LDLIBS='-lsocket -lnsl'
$ ./netminesweeperd     # server
$ ./xnetminesweeper     # client
```

これは、ネットワーク対戦型マインスイーパサーバ／クライアントプログラムであり、ネットワーククライアントにおいて X11 グラフィックスを利用しているものの、基本的に `sockets/{sans-fchatd,sans-fchatc}`と同じプログラムである。

ネットワークプログラミング：まとめ

echo, chat サーバ／クライアントの作成を通して：

- ソケット API の基礎
- UNIX, INET, INET6 アドレスファミリ（IPC, IPv4, IPv4/IPv6）における通信
- ソケットタイプ DGRAM, STREAM（TCP: Transmission Control Protocol, UDP: User Datagram Protocol）による通信
- 名前アドレス変換 `gethostbyname` ／ `gethostbyaddr`、`getaddrinfo` ／ `getnameinfo`
- INET, DGRAM におけるブロードキャスト
- fork, pthreads による並行サーバ、select による I/O の多重化
- fdopen による高水準入出力の利用

その他の重要な話題：

- `fcntl` 等による非ブロッキング I/O、`signal` によるシグナル駆動 I/O、`aio_read` 等による非同期 I/O
- マルチキャスト
- デーモンプロセスやスーパーサーバ (inetd,xinetd,etc)
- パフォーマンスやセキュリティに関するテクニック
- 名前アドレス変換と DNS とその設計とセキュリティと…

参考文献：

- 「TCP/IP によるネットワーク構築 Vol.III〜クライアントサーバプログラミングとアプリケーション（村井 純 他 訳、共立出版）」
- 「UNIX ネットワークプログラミング Vol.1〜ネットワーク API:ソケットと XTI（篠田 陽一 訳、ピアソン・エデュケーション）」…特にお勧め！
- 「Solaris 10 プログラミングインターフェースガイド 第８章 ソケットインターフェース（Sun Microsystems）」
  `http://docs.sun.com/app/docs/doc/817-4415/sockets-85885`
- 「Solaris 10 プログラミングインターフェースガイド 第９章 XTI と TLI によるプログラミング（Sun Microsystems）」
  `http://docs.sun.com/app/docs/doc/817-4415/tli-33281`