itojun 氏による、従来型 IPv4 クライアントプログラムと、デュアルスタック対応クライアントプログラムのサンプル

```
/*
 * client by gethostby* (IPv4 only)
 * by Jun-ichiro itojun Hagino.   in public domain.
 */

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

int
main(argc, argv)
        int argc;
        char **argv;
{
```

このプログラムでは、第一引数でホスト、第二引数でポートの指定を受け付ける。

そして、起動時に指定されたホストのポートにクライアントとして接続に行くというツールである。

```
/*
 * client by getaddrinfo (multi-protocol support)
 * by Jun-ichiro itojun Hagino.   in public domain.
 */

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

int
main(argc, argv)
        int argc;
        char **argv;
{
```

```
struct hostent *hp;
struct servent *sp;
unsigned long lport;
u_int16_t port;
char *ep;
struct sockaddr_in dst;
int dstlen;
ssize_t l;
int s;
char hbuf[INET_ADDRSTRLEN];
char buf[1024];

/* check the number of arguments */
if (argc != 3) {
        fprintf(stderr, "usage: test host port¥n");
        exit(1);
        /*NOTREACHED*/
}

/* resolve host name into binary */
hp = gethostbyname(argv[1]);
if (!hp) {
        fprintf(stderr, "%s: %s¥n", argv[1], hstrerror(h_errno));
```

gethostbyname を利用している
(旧式である)

```
struct addrinfo hints, *res, *res0;
ssize_t l;
int s;
char hbuf[NI_MAXHOST], sbuf[NI_MAXSERV];
char buf[1024];
int error;

/* check the number of arguments */
if (argc != 3) {
        fprintf(stderr, "usage: test host port¥n");
        exit(1);
        /*NOTREACHED*/
}

/* resolve address/port into sockaddr */
memset(&hints, 0, sizeof(hints));
hints.ai_socktype = SOCK_STREAM;
error = getaddrinfo(argv[1], argv[2], &hints, &res0);
```

```
                exit(1);
                /*NOTREACHED*/
        }
        if (hp->h_length != sizeof(dst.sin_addr)) {
                fprintf(stderr, "%s: unexpected address length¥n", argv[1]);
                exit(1);
                /*NOTREACHED*/
        }
        /* resolve port number into binary */
        sp = getservbyname(argv[2], "tcp");
        if (sp) {
                port = sp->s_port & 0xffff;
        } else {
                ep = NULL;
                errno = 0;
                lport = strtoul(argv[2], &ep, 10);
                if (!*argv[2] || errno || !ep || *ep) {
                        fprintf(stderr, "%s: no such service¥n", argv[2]);
                        exit(1);
                        /*NOTREACHED*/
                }
                if (lport & ~0xffff) {
                        fprintf(stderr, "%s: out of range¥n", argv[2]);
```

getservbyname を利用している
(旧式である)

```
        if (error) {
                fprintf(stderr, "%s %s: %s¥n", argv[1], argv[1],
                    gai_strerror(error));
                exit(1);
                /*NOTREACHED*/
        }
```

getaddrinfo を利用して一気に名前引きを行う。getaddrinfo は gethostbyname と getservbyname の機能を持っている上に、ホスト名から引けるアドレスを 1 回の関数コールで全て取得する。これにより、IPv4/IPv6 を問わず、複数の IP アドレスがリストの形式で一括して得られる。

不定数のアドレス情報が getaddrinfo 関数の中で確保されたメモリ空間に格納される。つまり、res0 が示す先のメモリオブジェクトは動的に確保されている。
このため、getaddrinfo 関数で得られた結果の利用が完了したら、そのメモリオブジェクトを解放する必要がある。それが freeaddrinfo()である。

```
                exit(1);
                /*NOTREACHED*/
        }

        port = htons(lport & 0xffff);
}
endservent();

/* try the first address only */
memset(&dst, 0, sizeof(dst));
dst.sin_family = AF_INET;
/* linux/Solaris does not need the following line */
dst.sin_len = sizeof(struct sockaddr_in);
memcpy(&dst.sin_addr, hp->h_addr, sizeof(dst.sin_addr));
dst.sin_port = port;
dstlen = sizeof(struct sockaddr_in);

s = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
if (s < 0) {
        perror("socket");
        exit(1);
        /*NOTREACHED*/
}
```

発見された接続先の、最初のものだけに接続に行く。しかも、ファミリは IPv4 固定である。
単一プロトコルの状況なら良いが、デュアルスタック環境においてはこのような実装は許されない。

inet_ntop は古い関数であり使用すべきではない。getnameinfo に変更すべき。

getaddrinfo の結果は、res にリスト形式で保存されている。
そのリストを辿りつつ、先頭から順番に適切なソケットを生成し、接続していく。

なお、このプログラムでは、接続の前にさらに getnameinfo でホスト名を取り直して、表示するようにしている。

```
/* try all the sockaddrs until connection goes successful */
for (res = res0; res; res = res->ai_next) {
        error = getnameinfo(res->ai_addr, res->ai_addrlen, hbuf,
            sizeof(hbuf), sbuf, sizeof(sbuf),
            NI_NUMERICHOST | NI_NUMERICSERV);
        if (error) {
                fprintf(stderr, "%s %s: %s¥n", argv[1], argv[1],
                    gai_strerror(error));
                continue;
        }
        fprintf(stderr, "trying %s port %s¥n", hbuf, sbuf);

        s = socket(res->ai_family, res->ai_socktype, res->ai_protocol);
        if (s < 0)
```

```
        inet_ntop(AF_INET, hp->h_addr, hbuf, sizeof(hbuf));
        fprintf(stderr, "trying %s port %u¥n", hbuf, ntohs(port));

        if (connect(s, (struct sockaddr *)&dst, dstlen) < 0) {
                perror("connect");
                exit(1);
                /*NOTREACHED*/
        }

        while ((l = read(s, buf, sizeof(buf))) > 0)
                write(STDOUT_FILENO, buf, l);
        close(s);
        exit(0);
        /*NOTREACHED*/
}
```

```
                        continue;

                if (connect(s, res->ai_addr, res->ai_addrlen) < 0) {
                        close(s);
                        s = -1;
                        continue;
                }

                while ((l = read(s, buf, sizeof(buf))) > 0)
                        write(STDOUT_FILENO, buf, l);
                close(s);

                exit(0);
                /*NOTREACHED*/
        }

        fprintf(stderr, "test: no destination to connect to¥n");
        exit(1);
        /*NOTREACHED*/
}
```

■itojun 氏による、inetd を利用した IPv4 専用サーバプログラムと、inetd を利用したデュアルスタック対応サーバプログラムのサンプル

```
/*
 * server invoked via inetd (IPv4 only)
 * by Jun-ichiro itojun Hagino.  in public domain.
 */

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <stdio.h>
#include <errno.h>
#include <unistd.h>
#include <string.h>
#include <arpa/inet.h>



int
main(argc, argv)
        int argc;
        char **argv;
{
```

inetd を利用することにより、サーバプログラムの通信処理は、標準入出力への読み書きとして実装できる。
つまり、サーバとして送受信処理を行う範囲では、IPv4 用のプログラムを IPv6 で利用しても問題ない。

ただし、inetd からローンチされるプログラムでは、ロギングなどの用途で、通信相手の IP アドレスなどを知る機能がある。

これらの機能を使う時に、IPv4 のみを前提としているか、デュアルスタックを考慮しているかが異なる。

```
/*
 * server invoked via inetd (multi-protocol support)
 * by Jun-ichiro itojun Hagino.  in public domain.
 */

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <stdio.h>
#include <errno.h>
#include <unistd.h>
#include <string.h>
#include <netdb.h>
#include <arpa/inet.h>

int
main(argc, argv)
        int argc;
        char **argv;
{
```

```
struct sockaddr_in from;
socklen_t fromlen;
char hbuf[INET_ADDRSTRLEN];

/* get the peer's address */
fromlen = sizeof(from);
if (getpeername(0, (struct sockaddr *)&from, &fromlen) < 0) {
        exit(1);
        /*NOTREACHED*/
}
if (from.sin_family != AF_INET ||
    fromlen != sizeof(struct sockaddr_in)) {
        exit(1);
        /*NOTREACHED*/
}

if (inet_ntop(AF_INET, &from.sin_addr, hbuf, sizeof(hbuf)) == NULL) {
        exit(1);
        /*NOTREACHED*/
}

write(0, "hello ", 6);
```

通信相手の情報を得る関数は getpeername である。これは sockaddr 構造体にも対応している比較的新しい API を持っている。

inet_ntop で、ネットワークフォーマットから文字列に変換している。
この inet_ntop / inet_pton は使うべきではない。

```
struct sockaddr_storage from;
socklen_t fromlen;
char hbuf[NI_MAXHOST];

/* get the peer's address */
fromlen = sizeof(from);
if (getpeername(0, (struct sockaddr *)&from, &fromlen) < 0) {
        exit(1);
        /*NOTREACHED*/
}

if (getnameinfo((struct sockaddr *)&from, fromlen, hbuf, sizeof(hbuf),
    NULL, 0, NI_NUMERICHOST) != 0) {
        exit(1);
        /*NOTREACHED*/
}

write(0, "hello ", 6);
write(0, hbuf, strlen(hbuf));
```

getpeername は sockaddr 構造体にも対応しており、デュアルスタックでもそのまま利用可能である。

IP アドレスを文字列化するのに getnameinfo を利用する。

```
        write(0, hbuf, strlen(hbuf));
        write(0, "¥n", 1);
        exit(0);
}
```

通信そのものは標準入出力への読み書きであるし、ソケットオープンなども不要である。このため IPv4 と IPv6 の違いはない。

```
        write(0, "¥n", 1);
        exit(0);
}
```

通信そのものは標準入出力への読み書きであるし、ソケットオープンなども不要である。このため IPv4 と IPv6 の違いはない。

■itojun 氏による、IPv4 専用サーバプログラムと、複数の socket を生成するデュアルスタック対応サーバプログラムのサンプル

```
/*
 * server with single listening socket (IPv4 only)
 * by Jun-ichiro itojun Hagino.  in public domain.
 */

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

```
/*
 * server with multiple listening socket based on getaddrinfo
 * (multi-protocol support)
 * by Jun-ichiro itojun Hagino.  in public domain.
 */

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

#define MAXSOCK    20
```

```
int
main(argc, argv)
        int argc;
        char **argv;
{
        struct servent *sp;
        unsigned long lport;
        u_int16_t port;
        char *ep;
        struct sockaddr_in serv;
        int servlen;
        struct sockaddr_in from;
        socklen_t fromlen;
        int s;
        int ls;
        char hbuf[INET_ADDRSTRLEN];

        if (argc != 2) {
                fprintf(stderr, "usage: test port¥n");
                exit(1);
                /*NOTREACHED*/
        }
        sp = getservbyname(argv[1], "tcp");
```

```
int
main(argc, argv)
        int argc;
        char **argv;
{
        struct addrinfo hints, *res, *res0;
        int error;
        struct sockaddr_storage from;
        socklen_t fromlen;
        int ls;
        int s[MAXSOCK];
        int smax;
        int sockmax;
        fd_set rfd, rfd0;
        int n;
        int i;
        char hbuf[NI_MAXHOST], sbuf[NI_MAXSERV];
#ifdef IPV6_V6ONLY
        const int on = 1;
#endif

        if (argc != 2) {
                fprintf(stderr, "usage: test port¥n");
```

```
        if (sp)
                port = sp->s_port & 0xffff;
        else {
                ep = NULL;
                errno = 0;
                lport = strtoul(argv[1], &ep, 10);
                if (!*argv[1] || errno || !ep || *ep) {
                        fprintf(stderr, "%s: no such service¥n", argv[1]);
                        exit(1);
                        /*NOTREACHED*/
                }
                if (lport & ~0xffff) {
                        fprintf(stderr, "%s: out of range¥n", argv[1]);
                        exit(1);
                        /*NOTREACHED*/
                }

                port = htons(lport & 0xffff);
        }
        endservent();

        memset(&serv, 0, sizeof(serv));
        serv.sin_family = AF_INET;
```

スイッチで与えられたポート番号の確認を行う。
getservbyname や gethostbyname はスレッドセーフではないので注意が必要。

```
                exit(1);
                /*NOTREACHED*/
        }

        memset(&hints, 0, sizeof(hints));
        hints.ai_socktype = SOCK_STREAM;
        hints.ai_flags = AI_PASSIVE;
        error = getaddrinfo(NULL, argv[1], &hints, &res0);
        if (error) {
                fprintf(stderr, "%s: %s¥n", argv[1], gai_strerror(error));
                exit(1);
                /*NOTREACHED*/
        }
```

サーバ側の指定されたポートで listen 可能なアドレスを getaddrinfo で取得する。ちなみに、ポートはこのプログラムの実行時スイッチパラメータから与えられている。
サーバ側の情報を得るためには、ヒントの ai_flags に AI_PASSIVE を指定する。

```
        /* linux/Solaris does not need the following line */
        serv.sin_len = sizeof(struct sockaddr_in);
        serv.sin_port = port;
        servlen = sizeof(struct sockaddr_in);

        s = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
        if (s < 0) {
                perror("socket");
                exit(1);
                /*NOTREACHED*/
        }

        if (bind(s, (struct sockaddr *)&serv, servlen) < 0) {
                perror("bind");
                exit(1);
                /*NOTREACHED*/
        }
        if (listen(s, 5) < 0) {
                perror("listen");
                exit(1);
                /*NOTREACHED*/
        }
```

sockaddr_in で、しかもファミリなど決めうちで socket をオープン、bind、listen している。
複数のプロトコルや複数の IP アドレスに接続される可能性のあるサーバではやってはいけないこと。

```
        smax = 0;
        sockmax = -1;
        for (res = res0; res && smax < MAXSOCK; res = res->ai_next) {
                s[smax] = socket(res->ai_family, res->ai_socktype,
                    res->ai_protocol);
                if (s[smax] < 0)
                        continue;

                /* avoid FD_SET overrun */
                if (s[smax] >= FD_SETSIZE) {
                        close(s[smax]);
                        s[smax] = -1;
                        continue;
                }

#ifdef IPV6_V6ONLY
                if (res->ai_family == AF_INET6 &&
                    setsockopt(s[smax], IPPROTO_IPV6, IPV6_V6ONLY, &on,
                    sizeof(on)) < 0) {
                        perror("bind");
                        s[smax] = -1;
                        continue;
```

得られたホストアドレス全てについて、socket 生成、setsockopt、bind、listen を行う。
このように、ファミリなどに応じて複数の socket を生成・管理・利用するのが正しいデュアルスタックプログラミングである。

```
        while (1) {
                fromlen = sizeof(from);
                ls = accept(s, (struct sockaddr
*)&from, &fromlen);
                if (ls < 0)
                        continue;
                if (from.sin_family != AF_INET ||
                    fromlen           !=
sizeof(struct sockaddr_in)) {
                        exit(1);
                        /*NOTREACHED*/
                }

                if (inet_ntop(AF_INET, &from.sin_addr, hbuf, sizeof(hbuf)) ==
                    NULL) {
                        exit(1);
                        /*NOTREACHED*/
                }

                write(ls, "hello ", 6);
                write(ls, hbuf, strlen(hbuf));
                write(ls, "¥n", 1);
                close(ls);
                }
#endif

                if (bind(s[smax], res->ai_addr, res->ai_addrlen) < 0) {
                        close(s[smax]);
                        s[smax] = -1;
                        continue;
                }
                if (listen(s[smax], 5) < 0) {
                        close(s[smax]);
                        s[smax] = -1;
                        continue;
                }

                error = getnameinfo(res->ai_addr, res->ai_addrlen, hbuf,
                    sizeof(hbuf), sbuf, sizeof(sbuf),
                    NI_NUMERICHOST | NI_NUMERICSERV);
                if (error) {
                        fprintf(stderr, "test: %s¥n", gai_strerror(error));
                        exit(1);
                        /*NOTREACHED*/
                }
                fprintf(stderr, "listen to %s %s¥n", hbuf, sbuf);
```

何も考えずに、開いた fd に対して accept を行い、通信を行う。

inet_ntop を使い、IPv4 アドレスを表示可能な形式に変換している。この辺の関数は一掃しなければならない。

従来通りの方式で、fd に対して read/write を行い、通信を完了する。

接続先の情報を得るために、getnameinfo を使って情報を取得している。

```
        }
        /*NOTREACHED*/
}
```

```
        if (s[smax] > sockmax)
                sockmax = s[smax];
        smax++;
}

if (smax == 0) {
        fprintf(stderr, "test: no socket to listen to¥n");
        exit(1);
        /*NOTREACHED*/
}

FD_ZERO(&rfd0);
for (i = 0; i < smax; i++)
        FD_SET(s[i], &rfd0);

while (1) {
        rfd = rfd0;
        n = select(sockmax + 1, &rfd, NULL, NULL, NULL);
        if (n < 0) {
                perror("select");
                exit(1);
                /*NOTREACHED*/
```

select 関数を使うため、fd_set を作成する。
listen している（待ち合わせしている）socket のデスクリプタを一覧にする。

それを select 関数に入れて、複数デスクリプタの待ち合わせを行う。

```
                }
                for (i = 0; i < smax; i++) {
                        if      (FD_ISSET(s[i],
&rfd)) {
                                fromlen = sizeof(from);
                                ls = accept(s[i], (struct sockaddr *)&from,
                                    &fromlen);
                                if (ls < 0)
                                        continue;
                                write(ls, "hello¥n", 6);
                                close(ls);
                        }
                }
        }


        /*NOTREACHED*/
}
```

FD_ISSET でセットされている fd に対して、accept を行う。
このようにして、複数の socket の fd を待ち合わせ、そのデスクリプタに対して、read/write を行う。

■itojun 氏による、IPv4/IPv6 選択型のサーバプログラムサンプル

```
/*
 * server with single listening socket (IPv4/v6 switchable)
 * by Jun-ichiro itojun Hagino.  in public domain.
 */

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

int
main(argc, argv)
          int argc;
          char **argv;
```

```
{
        struct addrinfo hints, *res;
        int error;
        struct sockaddr_storage from;
        socklen_t fromlen;
        int ls;
        int s;
        char hbuf[NI_MAXHOST], sbuf[NI_MAXSERV];
        int ch;
        int af = AF_INET6;
#ifdef IPV6_V6ONLY
        const int on = 1;
#endif

        while ((ch = getopt(argc, argv, "46")) != -1) {
                switch (ch) {
                case '4':
                        af = AF_INET;
                        break;
                case '6':
                        af = AF_INET6;
                        break;
                default:
```

```
                        fprintf(stderr, "usage: test [-46] port¥n");
                        exit(1);
                        /*NOTREACHED*/
                }
        }

        argc -= optind;
        argv += optind;

        if (argc != 1) {
                fprintf(stderr, "usage: test port¥n");
                exit(1);
                /*NOTREACHED*/
        }

        memset(&hints, 0, sizeof(hints));
        hints.ai_family = af;
        hints.ai_socktype = SOCK_STREAM;
        hints.ai_flags = AI_PASSIVE;
        error = getaddrinfo(NULL, argv[0], &hints, &res);
        if (error) {
                fprintf(stderr, "%s: %s¥n", argv[0], gai_strerror(error));
                exit(1);
```

スイッチパラメータの値に応じて、ファミリ値(AF_INET / AF_INET6)を選択する。

サーバとなる自分の IP アドレスを取得するためにも、getaddrinfo を利用する。

getaddrinfo を呼び出す際、ヒントの ai_flags を AI_PASSIVE と指定することにより、サーバ側の値が取得できる。
また、ファミリは先ほどスイッチパラメータに応じて設定した値とする。

これらの設定を行い、getaddrinfo を呼び出す。

```
                /*NOTREACHED*/
        }
        if (res->ai_next) {
                fprintf(stderr, "%s: multiple address returned¥n", argv[0]);
                exit(1);
                /*NOTREACHED*/
        }

        s = socket(res->ai_family, res->ai_socktype, res->ai_protocol);
        if (s < 0) {
                perror("socket");
                exit(1);
                /*NOTREACHED*/
        }

#ifdef IPV6_V6ONLY
        if (res->ai_family == AF_INET6 &&
            setsockopt(s, IPPROTO_IPV6, IPV6_V6ONLY, &on, sizeof(on)) < 0) {
                perror("bind");
                exit(1);
                /*NOTREACHED*/
        }
#endif
```

得られた値で socket をオープンする。本来は得られた結果がリストになっているので、そのリストの階数分だけ socket を生成すべきだが、今回はファミリを 1 個だけ選んでオープンする方式なので、基本的には 1 回だけのオープンで許される。このため、res のリストを追わず、リストの先頭の要素を参照することで済ましている。

IPv6_V6ONLY マクロで括られているこの部分は、v4 マップドアドレス機能を無効化する処理である。
v4 マップドアドレスは今回使用しないため、この指定をする。

```
if (bind(s, res->ai_addr, res->ai_addrlen) < 0) {
        perror("bind");
        exit(1);
        /*NOTREACHED*/
}
if (listen(s, 5) < 0) {
        perror("listen");
        exit(1);
        /*NOTREACHED*/
}

error = getnameinfo(res->ai_addr, res->ai_addrlen, hbuf,
    sizeof(hbuf), sbuf, sizeof(sbuf),
    NI_NUMERICHOST | NI_NUMERICSERV);
if (error) {
        fprintf(stderr, "test: %s¥n", gai_strerror(error));
        exit(1);
        /*NOTREACHED*/
}
fprintf(stderr, "listen to %s %s¥n", hbuf, sbuf);

while (1) {
```

オープンしたソケットを bind、listen する。
以降は通常通りの処理。

なお、下記では getnameinfo という、getaddrinfo の逆引版関数を使っている。この関数も IPv6 の環境に合わせた関数となっている。もし、プログラムの中で他の逆引関数を使っているのであればそれをやめ、この getnameinfo を利用するように修正すべきである。

```
                fromlen = sizeof(from);
                ls = accept(s, (struct sockaddr *)&from, &fromlen);
                if (ls < 0)
                        continue;
                write(ls, "hello¥n", 6);
                close(ls);
        }
        /*NOTREACHED*/
}
```