アプリケーションの IPv6 対応ガイドライン

基礎編 添付資料

アプリケーションの IPv6 化例示プログラム集

IPv6 普及･高度化推進協議会
IPv4/IPv6 共存 WG アプリケーションの IPv6 対応検討 SWG
2012 年 12 月 3 日

本文書について

本文書は IPv6 普及・高度化推進協議会 IPv4/IPv6 共存 WG アプリケーションの IPv6 対応検討 SWG で編集した文書である。
本文書は萩野純一郎氏（itojun 氏）による IPv6 ソケットプログラミングのサンプルプログラムを元にしている。itojun 氏が過去にパブリックドメインとして公開したサンプルプログラムについて、当該 SWG で下記の修正を行った。

- シングルスタックのサンプルとデュアルスタックのサンプルを比較しやすく横 2 列構成とし、サンプルソース内に適宜空行を入れた。
- 現在の記法に合わない部分を修正した。
- IPv4 シングルスタックプログラムを IPv4/IPv6 デュアルスタックプログラムに変更する際の要点を着色し、適宜解説文書を付与した。

このプログラムの詳細は ASCII 社刊 IPv6 ネットワークプログラミング 萩野純一郎（itojun）氏著 ISBN4-7561-4236-2 (978-4-7561-4236-8) において解説されている。
なお、本サンプル･解説におけるデュアルスタックとは、IPv4 と IPv6 両方の到達性や名前解決が可能であり、双方が共存して利用できる環境を示す。

■IPv6 対応クライアントプログラミング

従来型 IPv4 シングルスタックのクライアントプログラム（左）と、デュアルスタック対応クライアントプログラム（右）

```
/*
 * 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(int argc, char **argv)
{
        struct hostent *hp;
        struct servent *sp;
```

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>
#include <stdlib.h>

int
main(int argc, char **argv)
{
        struct addrinfo hints, *res, *res0;
```

```
unsigned long lport;
uint16_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));
        exit(1);
        /*NOTREACHED*/
```

gethostbyname を利用している
（旧式である）

```
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_family = AF_UNSPEC;
hints.ai_socktype = SOCK_STREAM;
error = getaddrinfo(argv[1], argv[2], &hints, &res0);
if (error) {
```

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

getservbyname を利用している
（旧式である）

```
                fprintf(stderr, "%s %s: %s¥n", argv[1], argv[2],
                    gai_strerror(error));
                exit(1);
                /*NOTREACHED*/
        }
```

デュアルスタックとするには *getaddrinfo* を利用して名前引きを行う。*getaddrinfo* は *gethostbyname* と *getservbyname* の機能を持っており、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。

```
		}

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

/* try the first address only */
memset(&dst, 0, sizeof(dst));
dst.sin_family = AF_INET;
memcpy(&dst.sin_addr, hp->h_addr, sizeof(dst.sin_addr));
dst.sin_port = port;
dstlen = sizeof(dst);

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

inet_ntoa(*(struct in_addr *)hp->h_addr);
fprintf(stderr, "trying %s port %u¥n", hbuf, ntohs(port));
```

発見された接続先の、最初のものだけに接続に行く。ソケットは 1 個、アドレスファミリは IPv4 である。デュアルスタックに対応するには修正が必要である。

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

前記の *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。
(*getnameinfo* の利用サンプルも兼ねている)

```
/* 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[2],
			    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)
			continue;

		if (connect(s, res->ai_addr, res->ai_addrlen) < 0) {
```

```
        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*/
}
```

```
                        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*/
}
```

■ inetd を使用した IPv6 対応サーバプログラミング

inetd を利用した IPv4 専用サーバプログラム（左）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（右）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 <stdlib.h>
#include <arpa/inet.h>



int
main(int argc, char **argv)
{
        struct sockaddr_in from;
        socklen_t fromlen;
```

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 から起動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で実装が異なる。

```
/*
 * 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 <stdlib.h>
#include <netdb.h>
#include <arpa/inet.h>


int
main(int argc, char **argv)
{
        struct sockaddr_storage from;
        socklen_t fromlen;
```

sockaddr_storage はシステムがサポートするあらゆるプロトコルの情報保存バッファとして利用できる。

```
        char *hbuf;

        /* 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*/
        }

        hbuf = inet_ntoa(from.sin_addr);

        write(0, "hello ", 6);
        write(0, hbuf, strlen(hbuf));
        write(0, "¥n", 1);
        exit(0);
}
```

sockaddr_in として定義された from 変数を引数で与えている。これは IPv4 であることを前提としている。

inet_ntoa 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。この関数は IPv4 に依存している。

通信そのものは標準入出力への読み書きであり、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。

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

sockaddr_storagege で定義された from 変数を引数として与えている。

使っている通信プロトコルに応じた形式で格納されている通信相手に関する情報を *getnameinfo* を利用して文字列化する。

通信そのものは標準入出力への読み書きであり、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。

■複数のソケットを使用した IPv6 対応サーバプログラミング

IPv4 専用サーバプログラム（左）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（右）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>









int
```

```
/*
 * 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
```

消費される Socket の最大数を 20 と定義している。

```
main(int argc, char **argv)
{
        struct servent *sp;
        unsigned long lport;
        uint16_t port;
        char *ep;
        struct sockaddr_in serv;
        int servlen;
        struct sockaddr_in from;
        socklen_t fromlen;
        int s;
        int ls;
        char *hbuf;

        if (argc != 2) {
                fprintf(stderr, "usage: test port¥n");
                exit(1);
                /*NOTREACHED*/
        }
        sp = getservbyname(argv[1], "tcp");
        if (sp)
                port = sp->s_port & 0xffff;
        else {
```

スイッチで与えられたポート番号の確認を行う。
getservbyname や *gethostbyname* はグローバル構造体を使用するためスレッドセーフではないので注意が必要。

```
main(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];
#ifdef IPV6_V6ONLY
        const int on = 1;
#endif

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

IPV6_V6ONLY については p.12 を参照

```
		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));
```

```
	memset(&hints, 0, sizeof(hints));
	hints.ai_socktype = SOCK_STREAM;
	hints.ai_flags = AI_PASSIVE;
	hints.ai_family = AF_UNSPEC;
	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 addrinfo 情報を *getaddrinfo* で得る。
hints の ai_family メンバに AF_UNSPEC を指定することで、全てのアドレスファミリを対象とする。
これに加えて第一引数を NULL にし、hints の ai_flags メンバに AI_PASSIVE を指定することで、OS が Listen 可能な全てのプロトコル・アドレスのための addrinfo のリストが生成され、res0 に返される。
なお、listen するポート番号は argv[1]としている。

```
        serv.sin_family = AF_INET;
        serv.sin_port = port;
        servlen = sizeof(serv);

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

sockaddr_in で、しかもアドレスファミリなど決めうちで *socket* を生成、*bind*、*listen* しており、IPv4 前提の構造となっている。

```
        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("setsockopt(IPV6_V6ONLY)");
                        close(s[smax]);
```

得られた addrinfo 全てについて、*socket* 生成、*setsockopt*、*bind*、*listen* を行う。
このように、プロトコルに応じて複数の socket を生成・管理・利用するのがデュアルスタックプログラミングである。

システム内で IPV6_V6ONLY マクロが定義されている場合、IPv4 Mapped IPv6 Address を除外する設定が可能である。今回のプログラムではそれを行うため *setsockopt* を行っている。

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




        while (1) {
                                s[smax] = -1;
                                continue;
                        }
#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;
                        }

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

                if (smax == 0) {
                        fprintf(stderr, "test: no socket to listen to¥n");
```

```
                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*/
                }

                hbuf = inet_ntoa(from.sin_addr);

                write(ls, "hello ", 6);
                write(ls, hbuf, strlen(hbuf));
                write(ls, "¥n", 1);
                close(ls);
        }
        /*NOTREACHED*/
}
```

開いた fd に対して *accept* を行い、通信を行う。

inet_ntoa を使い、IPv4 アドレスを表示可能な形式に変換している。

従来通りの方式で、デスクリプタに対して *write* を行い、通信を完了する。

```
        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*/
        }
        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;
```

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。

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 のデスクリプタを待ち合わせ、接続のあったデスクリプタに対して、*write* を行う。

```
                                        error = getnameinfo((struct sockaddr *)&from,
                                        fromlen, hbuf, sizeof(hbuf), NULL, 0,
                                        NI_NUMERICHOST);
                                        if (error) {
                                                exit(1);
                                                /*NOTREACHED*/
                                        }
                                        write(ls, "hello ", 6);
                                        write(ls, hbuf, strlen(hbuf));
                                        write(ls, "¥n", 1);
                                        close(ls);
                                }
                        }
                }
                                        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*/
}
```

■マルチプロセスによる IPv6 対応サーバプログラミング

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(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);
                /*NOTREACHED*/
        }
        if (res->ai_next) {
```

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

サーバとなる自分のプロトコルを取得するために *getaddrinfo* を利用する。

getaddrinfo を呼び出す際、第一引数を NULL にして、hints の ai_flags を AI_PASSIVE と指定することにより、サーバの対応プロトコルが取得できる。
また、アドレス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。

```
                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

        if (bind(s, res->ai_addr, res->ai_addrlen) < 0) {
                perror("bind");
```

得られた値で *socket* を生成する。通常は得られた結果がリストになっているので、そのリストの要素数分だけ socket を生成する。
しかし、当該プログラムではアドレスファミリを 1 個だけ選んでオープンするものとして、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、IPV6_V6ONLY マクロが定義されている（つまり利用可能である）環境において、IPv4 mapped address 機能を無効化するための処理である。
このプログラムでは IPv4 mapped address は今回使用しないため、この指定をする。

```
                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) {
                fromlen = sizeof(from);
                ls = accept(s, (struct sockaddr *)&from, &fromlen);
                if (ls < 0)
```

生成したソケットを *bind*、*listen* する。
以降は従来通りの処理。

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。

```
                        continue;
                write(ls, "hello¥n", 6);
                close(ls);
        }
        /*NOTREACHED*/
}
```