プロトコル非依存の
ソケットプログラミングの基礎

NTTサービスインテグレーション基盤研究所
加藤 淳也

2011年12月1日

© 2011 NTT Service Integration Laboratories

NTT

アウトライン

1. 本チュートリアルの目的
2. プロトコルに依存しないアプリケーション
3. IPv4を前提として設計されたアプリケーションのプロトコル非依存化（IPv6対応）
    1. TCPクライアント
    2. TCPサーバ
4. プロトコルに依存する処理
5. アドレス選択機構（RFC3484）によるアプリケーションの制御
6. IPv6対応アプリケーションに関する最新動向
7. まとめ

© 2011 NTT Service Integration Laboratories

1. 本チュートリアルの目的

© 2011 NTT Service Integration Laboratories

本チュートリアルの目的

- プロトコルに依存しないアプリケーションの基本的な設計を知る
  - RFC3493 “Basic Socket Interface Extension for IPv6” が解説する拡張APIの使用方法を理解する
  - アプリケーションをIPv6に対応させる際に、レイヤ3プロトコル（IPv4/IPv6）に依存しない構造とすることが推奨されている

- IPv4を前提として書かれたコードを、プロトコル非依存コードへ書き直すことで、IPv6対応アプリケーションの基本構造を理解する
  - TCPで通信を行うサーバとクライアントを題材とします

© 2011 NTT Service Integration Laboratories

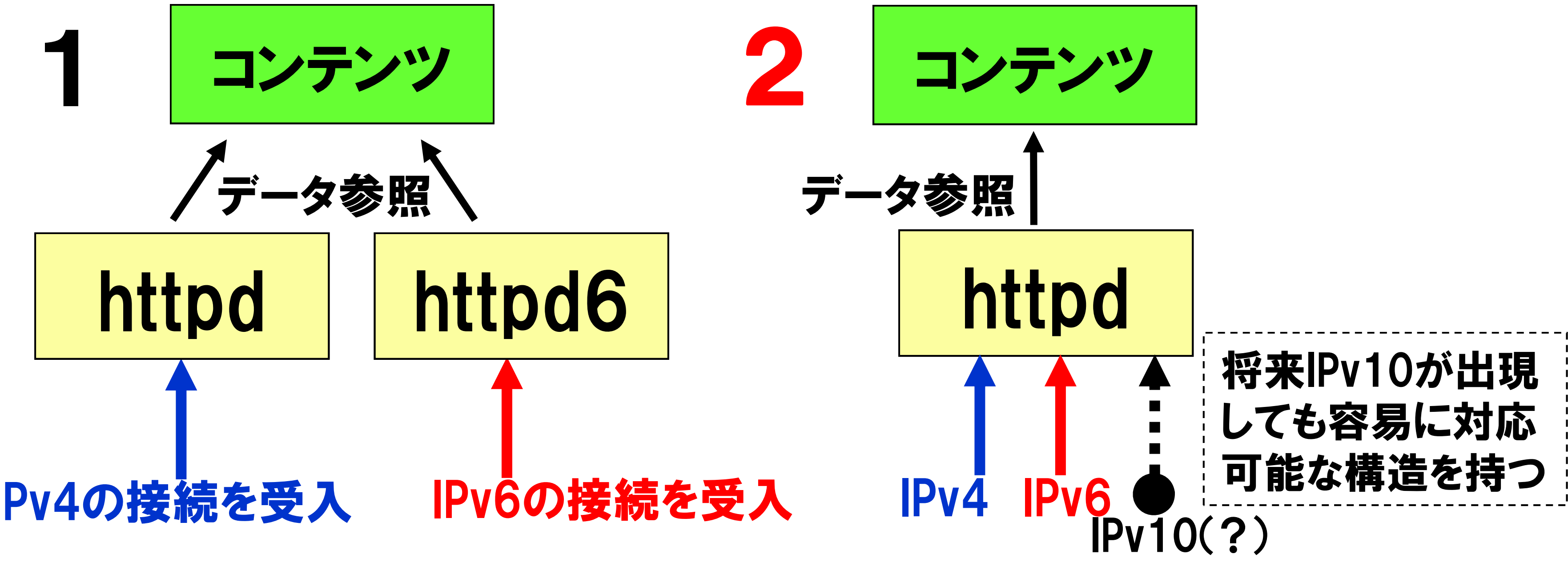
2. プロトコルに依存しないアプリケーション

© 2011 NTT Service Integration Laboratories

プロトコルに依存しないアプリケーション

アプリが使用するレイヤ3のプロトコルが異なっても同一のフローでネットワークI/Oを処理できるアプリケーションのこと

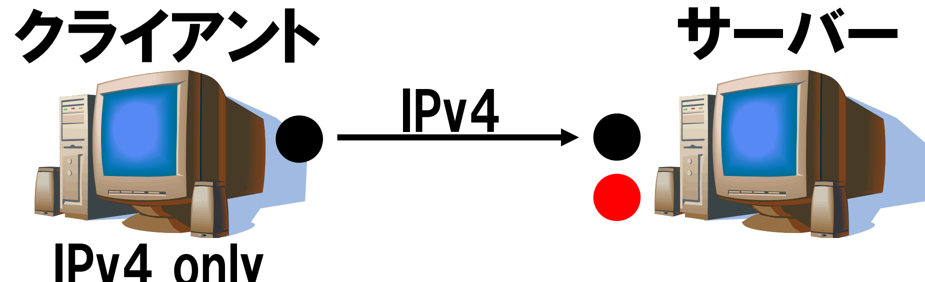
- IPv6対応のアプリケーションの多くは、IPv4, IPv6いずれでも接続できるものが一般的
  1. IPv6専用のアプリケーションと、IPv4専用のアプリケーションを同時に動作(並存)させる
  2. <u>レイヤ3のプロトコルに非依存なアプリケーション</u>
- IPv6対応のWebサーバの動作例

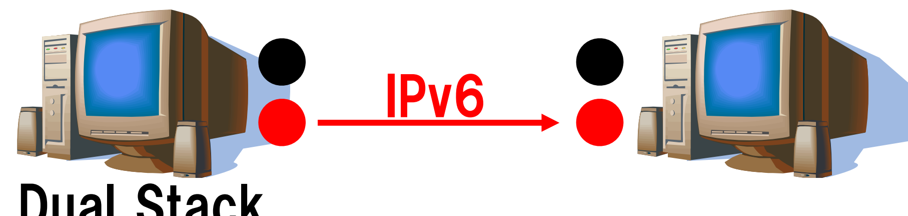


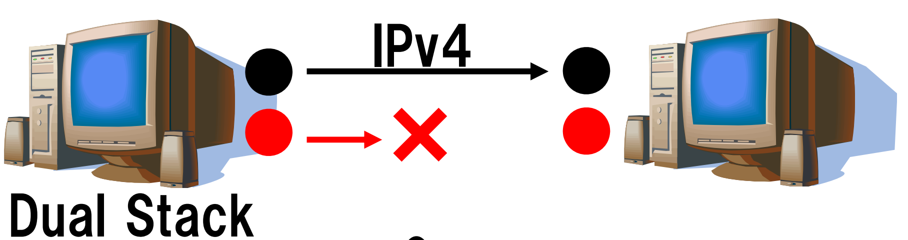
© 2011 NTT Service Integration Laboratories

NTT

プロトコルに依存しないアプリケーション（cont）







・
・
・多数の接続パタンが存在

- クライアントのプロトコルスタック
  - IPv6 only
  - IPv4 only
  - dual stack
- サーバのプロトコルスタック
  - IPv6 only
  - IPv4 only
  - dual stack
- ネットワークの状態
  - IPv6 ping OK
  - IPv4 ping OK

⇒ スタックの状態，ネットワークの状態の組み合わせによらず同一のコードでネットワークI/O処理を行う

© 2011 NTT Service Integration Laboratories

ソケットインターフェイス

- アプリケーションがプロトコルスタックを操作するためのI/F
  - 1983年 4.2BSDへTCP/IP搭載時に規定された
  - Windows Socket（WinSock）はBSDソケットから派生
- 仮想的な通信路のエンドポイント（ソケット）を提供し、アプリはソケットを指定するだけで通信可
  - 個別のプロトコル（TCPのSYNやACKなど）の通信手順を隠蔽
  - マルチプロトコルでの利用を前提とした設計
    - IPv4/IPv6だけでなく、IPX, NetBEUI, AppleTalk, OSI などIP以外のプロトコルを使うアプリケーションでも広く利用されるAPI

<table>
<tr><td colspan="5">アプリケーション</td><td></td></tr>
<tr><td colspan="5">ソケットレイヤ</td><td rowspan="3">OSカーネル</td></tr>
<tr><td>IPv4</td><td>IPv6</td><td>IPX</td><td>NetBEUI</td><td>AppleTalk</td></tr>
<tr><td colspan="5">デバイスドライバ</td></tr>
<tr><td colspan="5">ネットワークインターフェイス</td><td></td></tr>
</table>

© 2011 NTT Service Integration Laboratories

3. IPv4を前提として設計されたアプリケーションのプロトコル非依存化（IPv6対応）

© 2011 NTT Service Integration Laboratories

NTT

書き換えの題材 -TCPサーバとクライアント

- クライアントはサーバにTCPセッションを生成して、サーバから1行だけ文字列を受け取り、画面に表示する

■クライアント

tcp-client "サーバ名" ポート番号

DNS

■サーバ

tcp-server

名前解決
- DNS query
- DNS answer

コネクション開設
- TCP SYN
- TCP SYN+ACK
- TCP ACK

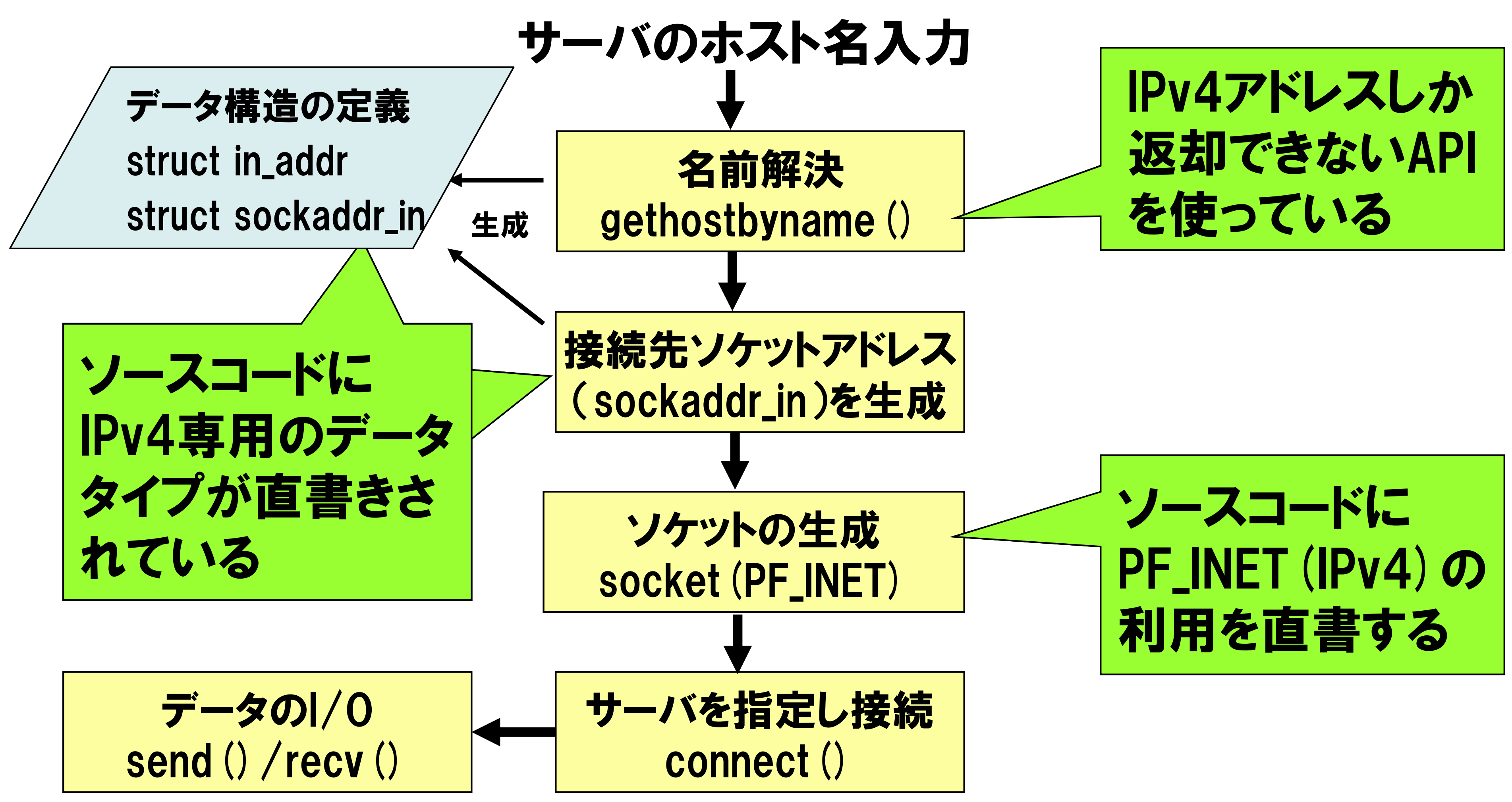
ネットワークI/O
- クライアントへデータ送信 "hello client!"

クライアントは受け取った文字列を画面に表示

コネクションのクローズ（FIN/ACK）

© 2011 NTT Service Integration Laboratories

IPv4の利用を前提としたTCPクライアント



© 2011 NTT Service Integration Laboratories

IPv4アプリケーション サンプルコード（TCPクライアント編）

```
/*
 * TCP echo クライアント IPv4バージョン
 */
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

int main(int argc, char *argv[])
{
    char *hostname;           /* サーバのホスト名 */
    char *servname;           /* ポート名 */
    struct hostent *hp;       /* IPv4アドレスの解決結果 */
    struct servent *sp;       /* ポート名の解決結果(ポート番号) */
    unsigned short port;      /* ポート番号を格納 */
    struct sockaddr_in sin4;  /* ソケットアドレス */
    int s;                    /* ソケットディスクリプタ */
    char buf[4096];           /* 受信文字列の格納バッファ */
    int n;

    /* コマンドライン引数のチェック */
    if (argc != 3) {
        fprintf(stderr, "usage: v6-tcp-client <hostname>
  <port>¥n");
        exit(1);
    }

    /* コマンドラインの第一引数(ホスト名)と第二引数(ポート名)をセット */
    hostname = argv[1];
    servname = argv[2];

    /* ホスト名からIPv4アドレスを解決 */
    if ((hp = gethostbyname(hostname)) == NULL) {
        fprintf(stderr, "cannot resolv hostname: %s¥n",
  hostname);
        exit(1);
    }
    /* ポート名からポート番号を解決 */
    if ((sp = getservbyname(servname, "tcp")) == NULL) {
        port = atoi(servname);
        if (port == 0) {
            fprintf(stderr, "cannot resolv servname: %s¥n",
                    servname);
            exit(1);
        }
        /* ネットワークバイトオーダーへ変換 */
        port = htons(port);
    } else
        port = sp->s_port;

    /* sockaddr_in 構造体を初期化 */
    memset(&sin4, 0, sizeof(struct sockaddr_in));
    /* アドレスファミリはIPv4を指定 */
    sin4.sin_family = AF_INET;
    /* ソケットアドレスの長さを設定 BSD系UNIXではコメントを外す */
/*  sin4.sin_len = sizeof(struct sockaddr_in); */
    /* ソケットアドレスにIPv4アドレス情報を設定 */
    memcpy(&sin4.sin_addr, hp->h_addr, sizeof(struct
  in_addr));
    /* ソケットアドレスにポート番号の情報を設定 */
    sin4.sin_port = port;

    /* IPv4ソケットを生成する */
    s = socket(AF_INET, SOCK_STREAM, 0);
    if (s < 0) {
        perror("socket");
        exit(1);
    }
    /* sin4で指定したサーバのポートへ接続 */
    if (connect(s, (struct sockaddr *)&sin4, sizeof(struct
  sockaddr_in)) < 0) {
        perror("connect");
        exit(1);
    }
    /* サーバから文字列を受け取りコンソールに出力 */
    while ((n = read(s, buf, sizeof(buf) - 1)) != 0) {
        buf[n] = '¥0';
        puts(buf);
    }
    close(s);
}
```


© 2011 NTT Service Integration Laboratories

IPアドレスを表すデータ構造，ソケットアドレス構造

■ IPv4アドレス(32bits)

```
struct in_addr {
    uint32_t s_addr;
};
```

■ IPv6アドレス(128bits)

```
struct in6_addr {
    uint8_t  u6_addr8[16];
};
```

■ IPv4通信のためのソケットアドレス

```
struct sockaddr_in {
    uint8_t          sin_len;
    uint8_t          sin_family;
    uint16_t         sin_port;
    struct in_addr   sin_addr;
    char             sin_zero[8];
};
```

■ IPv6通信のためのソケットアドレス

```
struct sockaddr_in6 {
    uint8_t          sin6_len;
    uint8_t          sin6_family;
    uint16_t         sin6_port;
    uint32_t         sin6_flowinfo;
    struct in6_addr  sin6_addr;
    uint32_t         sin6_scope_id;
};
```

■アドレスファミリ非依存のソケットアドレス

```
struct sockaddr {
    uint8_t          sa_len;
    uint8_t          sa_family;
    char             sa_data[14];
};
```

IPアドレスに依存するソケットアドレスのポインタをソケットレイヤに渡す場合、そのポインタをstruct sockaddr構造体にキャストして引き渡す。

© 2011 NTT Service Integration Laboratories

IPv6対応のための書き換え方針

- 基本的な方針
  - 既存のIPv4のコードに、IPv6のコードを安易に付け加えない
    - （既存のIPv4接続処理）＋（追加するIPv6接続処理）
      ＋（フォールバック処理）

    IPv6とIPv4接続を切り替えるフォールバック処理の追加も必要なため、コード量は2倍以上になる

- 新たなリゾルバ getaddrinfo（3） を使用して、IPv4/IPv6以外のプロトコルも同一の処理で扱えるよう書き換える

RFC3493, “Basic Socket Interface Extension for IPv6”が推奨する作法で書くと、

IPv4 onlyのプログラムも簡潔なコードで書けます

© 2011 NTT Service Integration Laboratories

プロトコル非依存のTCPクライアントの基本

ホスト名・ポート名・L4プロトコル(TCP/UDP)を指定

名前解決
getaddrinfo()

生成

(候補 ai1)サーバの
(候補 ai2)サーバの
(候補 ai3)サーバの
ソケットアドレス

接続候補のリストを取得

リストに残りあり？

NO

どの候補にも接続不可のときは接続エラー

YES

ソケットの生成
socket(ai_family)

サーバを指定しコネクト
connect(ai_addr)

接続成功

YES

データのI/O
send()/recv()

NO

リストの次の項目を試す

プロトコルに関する情報はすべてAPIが返却したものをそのまま使う
ハードコーディングをしない

© 2011 NTT Service Integration Laboratories

NTT 新しい名前解決API getaddrinfo (3) 関数

- 入力
  - ホスト名　文字列
  - ポート番号　文字列
  - L3プロトコル（問わない, IPv4, or IPv6）
  - 通信タイプ（ストリーム [TCP], データグラム [UDP], or RAW）
- 出力
  - ソケットアドレスと接続に必要な付加情報の**リスト**
  - 接続の候補となりうるソケットアドレスが列挙される
- IPv4向けAPIとの違いは

```
getaddrinfo() ≒ gethostbyname()        /* ホスト名解決 */
                + getservbyname()       /* ポート名解決 */
                + ソケットアドレスの初期化
                + バイトオーダの隠蔽
```

- IEEE Std. 1003.1-2001 (POSIX)が規定する標準APIのため多くのプラットフォームで利用可能

© 2011 NTT Service Integration Laboratories

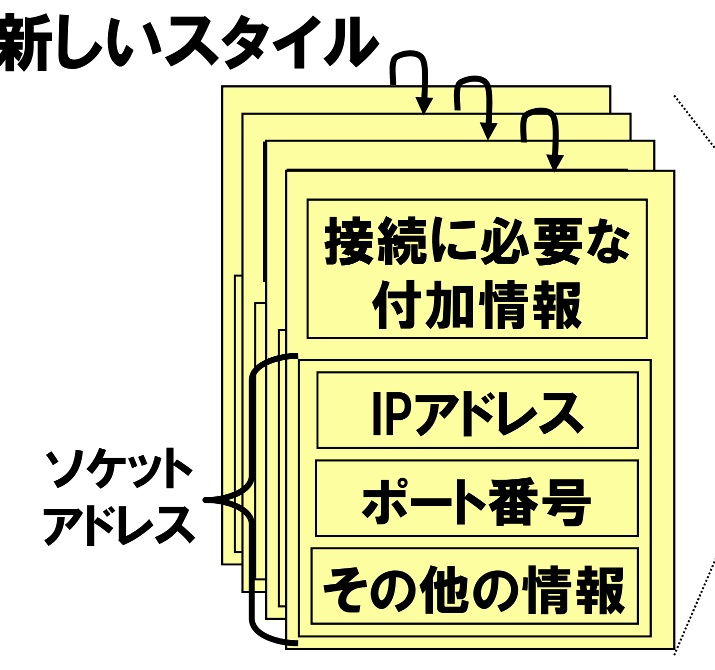
getaddrinfo (3) が生成する構造

従来のスタイル

ソケットアドレス
- IPv4アドレス
- ポート番号
- その他の情報

gethostbyname ()

APIが生成する情報はIPv4アドレスだけで残りの部分はプログラマが組み立てる



getaddrinfo ()

IPv4/IPv6どちらのアドレスファミリに対してもAPIがソケットアドレス全体を組み立てる。接続に必要な付加情報も提供

© 2011 NTT Service Integration Laboratories

addrinfo 構造体のデータ構造の役割

1. getaddrinfo () の振舞いを決める「ヒント」を格納する
2. ホスト名解決の結果を格納する

addrinfoのデータ構造

addrinfo
ai_flags
ai_family
ai_socktype
ai_protocol
ai_addrlen
ai_canonical（ポインタ）
ai_addr（ポインタ）
ai_next（ポインタ）

ai_flags:

```
AI_PASSIVE     /* サーバ用のソケットアドレスを生成 */
AI_CANONNAME   /* 別名を格納する */
AI_NUMERICHOST /* ホスト名は生アドレスで与えられる */
```

ai_family:

```
PF_UNSPEC   /* IPv6, IPv4 両方を対象とする */
PF_INET     /* IPv4のみ対象とする */
PF_INET6    /* IPv6のみ対象とする */
```

ai_socktype:

```
SOCK_STREAM  /* socketの第2引数 */
SOCK_DGRAM   /* TCP or UDP or any other */
```

```
/* wwwのポートhttp (80) にTCPで接続したい */
/* IPv4, IPv6は問わない */
memset(&hints, 0, sizeof(hints));
hints.ai_family = PF_UNSPEC;
hints.ai_socktype = SOCK_STREAM;
/* 名前解決 */
g = getaddrinfo("www", "http", &hints, &ai0);
```

（memset ～ hints.ai_socktype：ヒント情報生成）

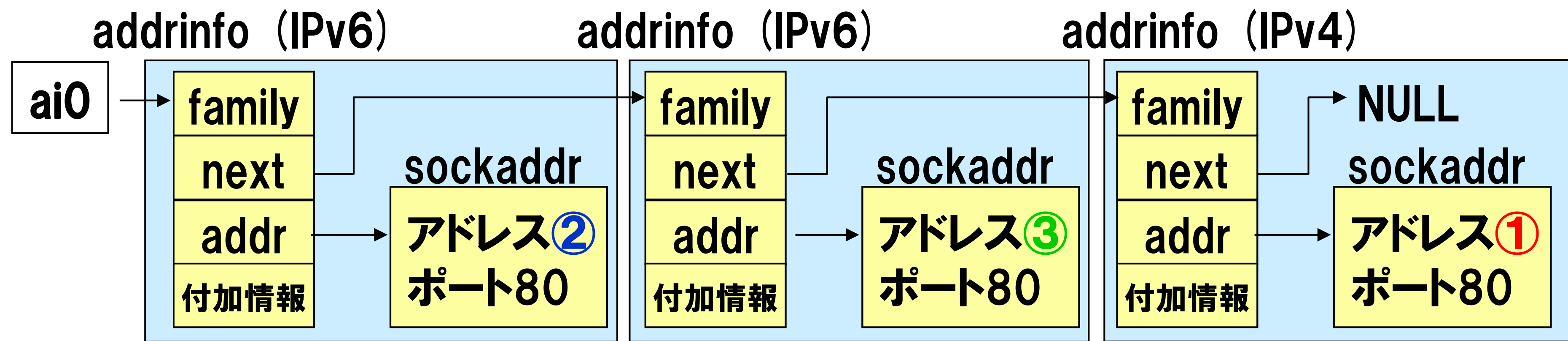
© 2011 NTT Service Integration Laboratories

異なるプロトコルのソケットアドレスの扱い方
(2) getaddrinfo () からの出力

・Webサーバに下記のIPアドレスが付与されている場合

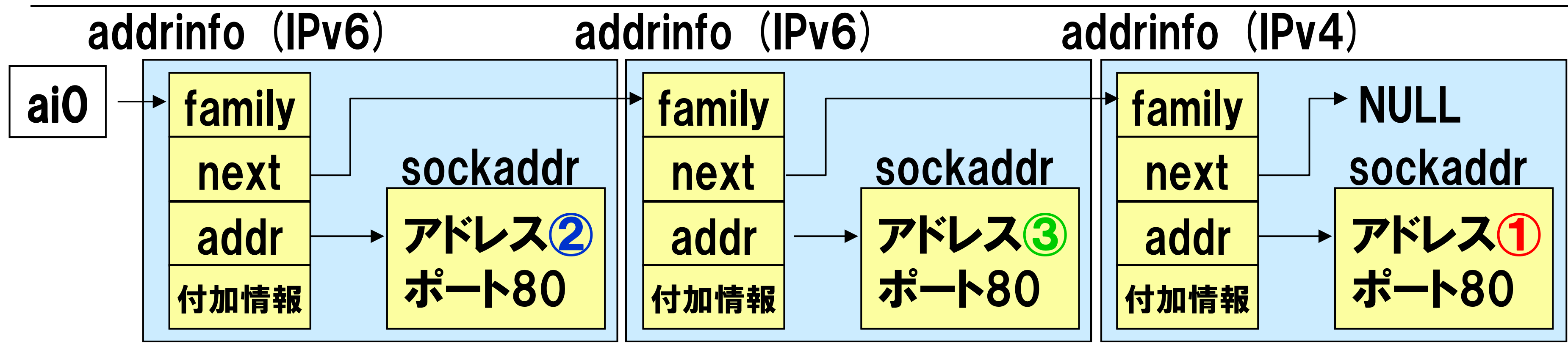
```
------------------------------------------------------------
www.example.com IN A     192.16.178.80  ・・・① IPv4
                IN AAAA  2001:db8::80  ・・・② IPv6
                IN AAAA  2001:db8:aaaa:80  ・・・③ IPv6
------------------------------------------------------------
```

・ai0を始点として、IPアドレスの数だけ連結リストが返却される
・多くのOSの標準の設定ではアドレスファミリごとにソートされる
1. IPv6アドレスのリスト
2. IPv4アドレスのリスト



© 2011 NTT Service Integration Laboratories

アドレスリストを活用したTCPコネクションの開設法



- ai0からnextを順にたどり、成功するまで試行を続ける
- すべてのaddrinfoが失敗したら接続失敗
- IPv6の接続が失敗したら、IPv4に自動的にフォールバックする動作が可能

```
for (ai = ai0; ai; ai = ai->ai_next) {
  s = socket(ai->ai_family,
             ai->ai_socktype, ai->ai_protocol);
  conenct(s, ai->ai_addr, ai->ai_addrlen); /* 接続試行 */
  if (connect()が失敗) {
    s = -1;
    continue; /* アドレスリストの次の項目の接続をトライ */
  }
  break;
}
if (s < 0) { 接続失敗; } /* すべてのアドレスに対して接続できなかった */
```

ソースコードにIPv4/IPv6に依存するコードが埋め込まれていない

© 2011 NTT Service Integration Laboratories

プロトコル非依存に書き換えたアプリケーション
サンプルコード(TCPクライアント編)

```
/*
 * サーバに接続し受信した文字列を表示するクライアント
 * プロトコル依存しないバージョン
 */
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>

int main(int argc, char *argv[])
{
    int s;                    /* ソケットディスクリプタ */
    struct addrinfo hints;  /* getaddrinfoのヒント情報 */
    struct addrinfo *ai0;   /* addrinfoリストの先頭の要素*/
    struct addrinfo *ai;    /* 処理中のaddrinfoリストの要素 */
    int g;                    /* 名前解決の結果 */
    char *nodename, *servname;
    char buf[4096];          /* 受信文字列の格納バッファ */
    int n;

    /* コマンドライン引数のチェック */
     if (argc != 3) {
          fprintf(stderr,
                  "usage: dual-tcp-client <hostname>
  <port>¥n");
          exit(1);
     }

    /* コマンドラインの第一引数(ホスト名)と第二引数(ポート名)をセット */
    nodename = argv[1];
    servname = argv[2];

    /* ヒント情報の初期化 */
    memset(&hints, 0, sizeof(hints));
    /* プロトコルを指定しない */
    hints.ai_family = PF_UNSPEC;
    /* ストリーム型(TCP)による通信を指定 */
    hints.ai_socktype = SOCK_STREAM;

    /* サーバ名とポート名の解決を行い、addrinfoリスト(ai0)を取得 */
    g = getaddrinfo(nodename, servname, &hints, &ai0);
    if (g) {
         fprintf(stderr, "%s", gai_strerror(g));
         exit(1);
    }
    /* addrinfoリストの要素を先頭から接続できるまで順に試行する */
    for (ai = ai0; ai; ai = ai->ai_next) {
        /* ソケットの生成 */
        s = socket(ai->ai_family, ai->ai_socktype,
                   ai->ai_protocol);

        /* ソケットの生成に失敗したらリストの次の項目を試す */
        if (s < 0)
            continue;

        /* ソケットが生成できたらサーバへ接続を試みる */
        /* 失敗したらリストの次の項目を試す */
        if (connect(s, ai->ai_addr, ai->ai_addrlen) < 0) {
            close(s);
            s = -1;
            continue;
        }

        /* ここではすでに接続に成功しているので、forループを脱出 */
        break;
    }
    /* リストのすべてのアドレスへの接続が失敗していないかチェック */
    if (s < 0) {
        fprintf(stderr, "cannot connect %s¥n", nodename);
        exit(1);
    }

    /* サーバから文字列を受け取りコンソールに出力 */
    while ((n = read(s, buf, sizeof(buf) - 1)) != 0) {
        buf[n] = '¥0';
        puts(buf);
    }
    close(s);
}
```


© 2011 NTT Service Integration Laboratories

IPv4の利用を前提としたTCPサーバ

データ構造の定義
struct in_addr
struct sockaddr_in

生成

サーバのソケットアドレス（sockaddr_in）を初期化

ソケットの生成
socket (PF_INET)

待受ポート番号を割付
bind ()

クライアントから接続を待受 listen ()

クライアントから接続を受け入れる accept ()

サーバ・クライアント間でデータのI/O recv/send ()

ソースコードにIPv4専用のデータタイプが直書きされてしまっている

ソースコードにPF_INET (IPv4) の利用が直書きされている

IPv4アドレスを前提としたソケットアドレス構造体を使用して待ち受けポート番号を指定

© 2011 NTT Service Integration Laboratories

IPv4に依存するコードを含んだTCPサーバ（1/2）

```
/*
 * 接続してきたクライアントに対して文字列を送信するサーバ
 * IPv4バージョン
 */
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <unistd.h>

int main(int argc, char *argv[])
{
    char *servname;          /* ポート名 */
    struct servent *sp;      /* ポート名の解決結果(ポート番号) */
    unsigned short port;     /* ポート番号を格納 */
    struct sockaddr_in sin4; /* ソケットアドレス */
    int s;                   /* ソケット */
    fd_set rfd, rfd0;

    /* コマンドラインの引数のチェック */
    if (argc != 2) {
        fprintf(stderr, "usage: v4-tcp-server <port>¥n");
        exit(1);
    }

    /* コマンドラインの第一引数(ポート名)をセット */
    servname = argv[1];

    /* ポート名からポート番号を解決 */
    if ((sp = getservbyname(servname, "tcp")) == NULL) {
        port = atoi(servname);
        if (port == 0) {
            fprintf(stderr, "cannot resolv servname: %s¥n",
   servname);
        exit(1);
        }
        /* ネットワークバイトオーダーへ変換 */
        port = htons(port);
    } else
        port = sp->s_port;

    /* sockaddr_in 構造体を初期化 */
    memset(&sin4, 0, sizeof(struct sockaddr_in));
    /* アドレスファミリとしてIPv4を指定 */
    sin4.sin_family = AF_INET;
    /* ソケットアドレスの長さを設定 BSD系UNIXではコメントを外す */
/*  sin4.sin_len = sizeof(struct sockaddr_in); */
    /* IPv4アドレスを設定(バイトオーダに注意)*/
    sin4.sin_addr.s_addr = htonl(INADDR_ANY);/* ポート番号設定 */
    sin4.sin_port = port;

    /* IPv4ソケットを生成する */
    s = socket(AF_INET, SOCK_STREAM, 0);
    if (s < 0) {
        perror("socket");
        exit(1);
    }

    /* ポート番号をソケットに結びつける */
    if (bind(s, (struct sockaddr *)&sin4, sizeof(struct
  sockaddr_in)) < 0) {
        perror("bind");
        close(s);
        exit(1);
    }

    /* クライアントからの接続の要求へ待機する */
    if (listen(s, 5) < 0) {
        perror("listen");
        close(s);
        exit(1);
    }

    /* ソケットsへのクライアントからの接続要求をselectにより監視 */
    FD_ZERO(&rfd0);
    FD_SET(s, &rfd0);

    /* サーバとのデータのI/O */
    while (1) {
        int as;
        struct sockaddr_in csin4; /* クライアントのソケットアドレス */
        int len;                  /* ソケットアドレスの長さ */
```


© 2011 NTT Service Integration Laboratories

IPv4に依存するコードを含んだTCPサーバ（2/2）

```
        int n;
        int pid;

    /* クライアントから接続が来るまでselectで待つ */
        rfd = rfd0;
        n = select(FD_SETSIZE, &rfd, NULL, NULL, NULL);
        if (n < 0) {
            printf("n = %d¥n", n);
            perror("select");
            close(s);
            exit(1);
        }

        /* クライアントからの接続を受け入れる */
        /* クライアントを示すソケットアドレスがcsin4に書き込まれる */
        as = accept(s, (struct sockaddr *)&csin4, &len);
        if (as < 0) {
            perror("accept");
            close(s);
            exit(1);
        }

        /* サブプロセスを生成し、クライアントとのI/Oを処理させる */
        if ((pid = fork()) < 0) {
            perror("fork");
            exit(1);
        }

        if (pid == 0) {
            /* サブプロセスの処理 */
            char *text = "send string: ¥"hello client!¥"¥n";
            write(as, text, strlen(text));
            close(as);
            exit(0);
        } else
            close(as);
    }
}
```


© 2011 NTT Service Integration Laboratories

・本チュートリアルでのサーバは同一のコードで、クライアントからのIPv4, IPv6接続のどちらでも受け入れる

＝ IPv4, IPv6両者のソケットを開いて待ち受ける

［比較］クライアントが通信に使うソケットは

１つだけ

接続が成功するまでアドレスのリストを順に試行するが、成功した時点で以降の試行は打ち切っていた

© 2011 NTT Service Integration Laboratories

AI_PASSIVEフラグつきのgetaddrinfo (3)

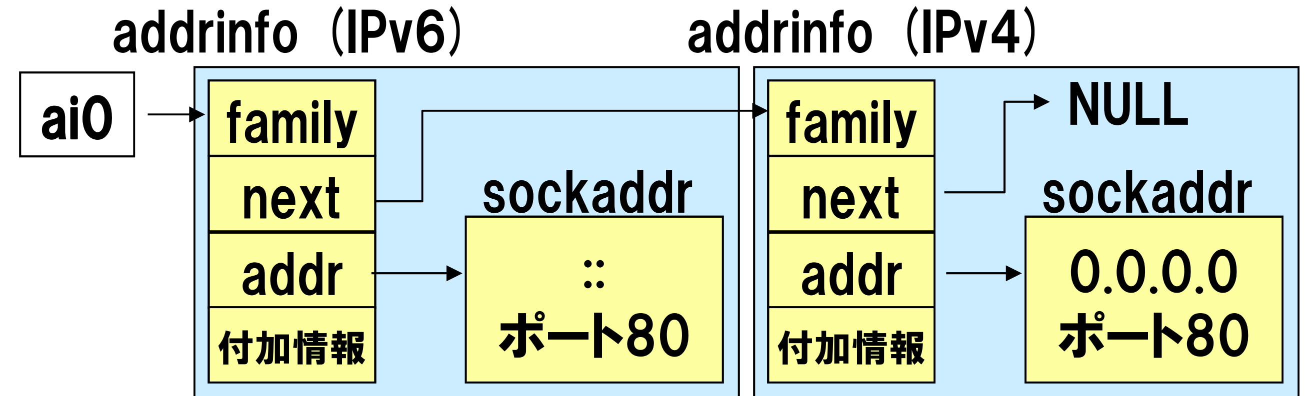
- ヒント情報にAI_PASSIVEを設定すると、名前解決は行わず、接続待ち受けのためのアドレス情報を生成
  - OSのプロトコル数に応じて、addrinfoのリストを生成
  - 通常、プロトコルを指定しない場合 IPv6, IPv4 の2項目

```
memset(&hints, 0, sizeof(hints));
hints.ai_family = PF_UNSPEC;
hints.ai_flags = AI_PASSIVE;
hints.ai_socktype = SOCK_STREAM;

g = getaddrinfo(NULL, "http", &hints, &ai0);
```


© 2011 NTT Service Integration Laboratories

TCPサーバの基本ロジック



すべてのプロトコル対してソケットを生成し待受ける(listen)

```
int s[MAXSOCKS];
for (smax = 0, ai = ai0; ai; ai = ai->ai_next) {
    s[smax] = socket(ai->ai_family, ai->ai_socktype,
                                        ai->ai_protocol);
    if (bind(s[smax], ai->addr, ai->addrlen) < 0)
        continue; /* bindに失敗したら次のプロトコルを試す */
    if (listen(s[smax], 5) < 0)
        continue; /* listenに失敗したら次のプロトコルを試す */
    smax++;
}
```


© 2011 NTT Service Integration Laboratories

クライアントからの接続受け入れ処理

- クライアントからの接続が到着するまでIPv4, IPv6どちらで来るかは不明
  - IPv4, IPv6　2つのソケットを開いて待受ける時、どちらかに接続が来たら受入れ処理を開始
  - 「どちらか」をselect () により判定する

- select (2) システムコールの役割
  - ソケットの集合を渡し、いずれか一つが読込み可能になるまで待つ
  - listen () により待ち受けているソケットをselect () に渡し、クライアントからの接続があるまで待つ

※接続受入れを多重化する方法は、マルチスレッド, 非同期I/O, ポーリングなど他の手法も存在するが、本チュートリアルではselect () システムコールを紹介する

© 2011 NTT Service Integration Laboratories

クライアントからの接続を受け入れるaccpet()

```
•int accept(int s,/* IN listen()で待つソケット */
            struct sockaddr *saddr, /* OUT */
            int *len                /* OUT */)
        /* OUTはaccept()が情報を渡すという意味 */
```

- 接続要求が来たクライアントとの通信(I/O)用ソケットを返す
- 接続してきたクライアントを示すソケットアドレスを返す
  - saddr には接続に使用されたプロトコルのソケットアドレスが設定される
    - クライアントがIPv4の通信をした場合 struct sockaddr_in
    - クライアントがIPv6の通信をした場合 struct sockaddr_in6
- プログラマは、IPv4, IPv6どちらの接続であってもsaddrにソケットアドレスが格納できるように、saddrに十分な大きさの領域を確保しておく必要がある

© 2011 NTT Service Integration Laboratories

２つの構造体の比較
sockaddr_in（IPv4）, sockaddr_in6（IPv6）

32ビット（４バイト）	
sin_famliy	sin_port
sin_addr	
pading [8]	

32ビット（４バイト）	
sin6_famliy	sin6_port
sin6_flowinfo	
sin6_addr	
sin6_scope_id	

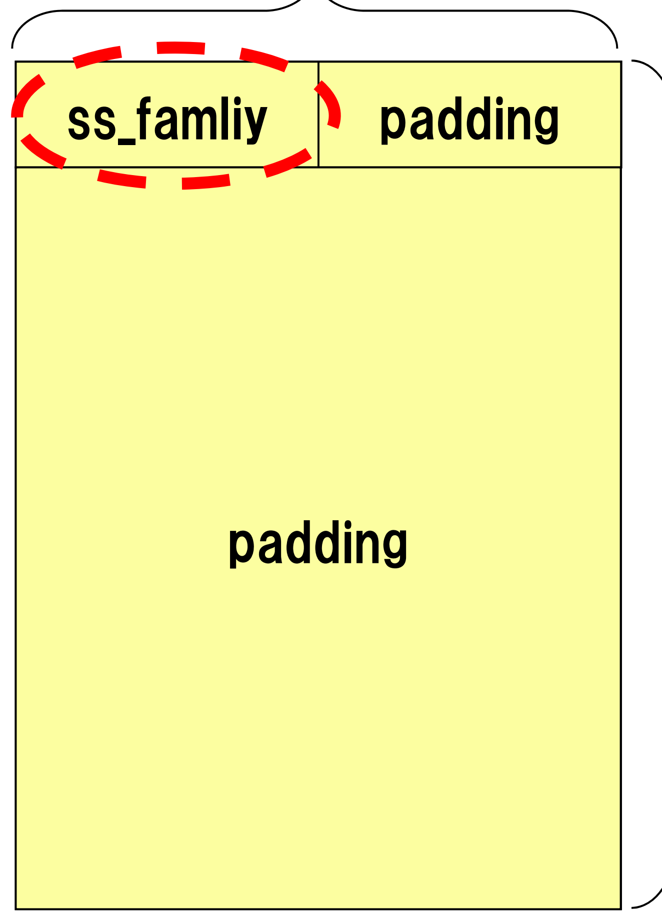
sizeof (struct sockaddr_in) sizeof (struct sockaddr_in6)

© 2011 NTT Service Integration Laboratories

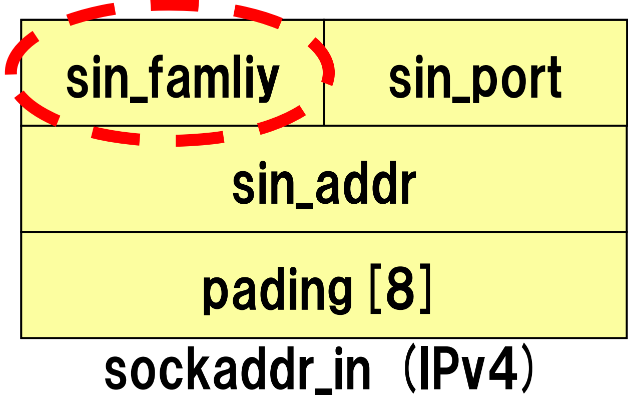
処理系が扱う任意のソケットアドレスが格納可能な sockaddr_storage 構造体

プロトコルを示すss_familyは他のソケットアドレスとオフセットを維持する

32ビット(４バイト)



sockaddr_storage



sockaddr_in （IPv4）

その処理系で最も大きなソケットアドレスが格納できるサイズ

sockaddr_storageは任意のソケットアドレスを格納できる「入れ物」。通信プロトコルが実行時まで決定しない場合に利用する

© 2011 NTT Service Integration Laboratories

select () とaccept () による接続処理の多重化

```
struct sockaddr_storage ss;
int sslen = sizeof(ss);

while (1) {
   /* IPv4, IPv6のいずれかの接続が来るまで待つ */
   select(FD_SETSIZE, &rfd, NULL, NULL, NULL);
   s = 接続が来たソケットをrfdから取り出す;

   /* 接続を受け入れる。 ss にはクライアントの情報が書き込まれる */
   as = accept(s,
               (struct sockaddr *)&ss,
               &sslen);

   スレッド or サブプロセス生成(as);
}
```

IPv4, IPv6どちらのソケットアドレスでもすっぽり格納できる領域を確保

```
スレッド or サブプロセス(int as){
   ソケットas に対してI/Oを行う
   recv(as, ...);
   send(as, ...);
}
```


© 2011 NTT Service Integration Laboratories

プロトコル非依存に書き換えたTCPサーバ（1/2）

```
/*
 * 接続してきたクライアントに対して文字列を送信するサーバ
 * IPv6/IPv4デュアルバージョン
 */
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

int main(int argc, char *argv[])
{
    int s[64];                /* クライアントと通信を行うソケット */
    struct addrinfo hints;    /* getaddrinfoのヒント情報 */
    struct addrinfo *ai0;     /* addrinfoリストの先頭の要素*/
    struct addrinfo *ai;      /* 処理中のaddrinfoリストの要素 */
    int g;                    /* getaddrinfoの処理結果 */
    char *servname;           /* ポート名 */
    int smax;                 /* 接続を受け入れるプロトコルの数 */
    fd_set rfd, rfd0;
    int i;

    /** コマンドライン引数のチェック */
    if (argc != 2) {
        fprintf(stderr, "usage: dual-tcp-server
  <port>¥n");
        exit(1);
    }
    /** コマンドラインの第一引数(ポート名)をセット */
    servname = argv[1];

    /** ヒント情報の初期化 */
    memset(&hints, 0, sizeof(hints));
    /* プロトコルは指定しない: IPv4 or IPv6 */
    hints.ai_family = PF_UNSPEC;
    /* ストリーム型(TCP)による通信を指定 */
    hints.ai_socktype = SOCK_STREAM;
    /* PASSIVE型のソケット生成を指定 */
    hints.ai_flags = AI_PASSIVE;

    /* 名前とポート名の解決 & 接続候補のリストを取得 */
    g = getaddrinfo(NULL, servname, &hints, &ai0);
    if (g) {
        fprintf(stderr, "%s", gai_strerror(g));
        exit(1);
    }

    smax = 0;
    /** addrinfoリストの要素を先頭から接続できるまで順に試行する */
    for (ai = ai0; ai; ai = ai->ai_next) {
        /* ソケットの生成 */
        s[smax] = socket(ai->ai_family, ai->ai_socktype,
ai->ai_protocol);

         /* ソケットの生成に失敗したらリストの次の項目を試す */
         if (s[smax] < 0)
            continue;
        /* ポート番号をソケットに結びつける */
        if (bind(s[smax], ai->ai_addr, ai->ai_addrlen) <
0) {
            perror("bind");
            close(s[smax]);
            continue;
        }

        /* クライアントからの接続の要求へ待機する */
        if (listen(s[smax], 5) < 0) {
            perror("listen");
            close(s[smax]);
            continue;
        }

        smax++;
    }
    /** forループで待受けソケットを生成できない場合はエラーとする */
    if (smax == 0) {
        fprintf(stderr, "cannot create server socket¥n");
        exit(1);
    }
    /** s[xx]へのクライアントからの接続要求をselectにより監視 */
    FD_ZERO(&rfd0);
```


© 2011 NTT Service Integration Laboratories

NTT プロトコル非依存に書き換えたTCPサーバ（2/2）

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

        /** クライアントとのI/O */
        while (1) {
            int as;
            struct sockaddr_storage ss; /* クライアントのソケット
アドレス */
            int sslen;                  /* ソケットアドレスの長さ */
            int n;
            int pid;

            /* クライアントが接続してくるまでselect()で待つ */
            rfd = rfd0;
            n = select(smax + 1, &rfd, NULL, NULL, NULL);
            if (n < 0) {
                perror("select");
                exit(1);
            }

            /* 接続を受け付けたソケットを求める */
            for (i = 0; i < smax; ++i) {
                if (FD_ISSET(s[i], &rfd))
                    break;
            }

            /* クライアントからの接続を受け入れる際に
               クライアントを示すソケットアドレスがssに書き込まれている */
            as = accept(s[i],
                        (struct sockaddr *)&ss, &sslen);
            if (as < 0) {
                perror("accept");
                exit(1);
            }

            /* サブプロセスを生成し、クライアントとのI/Oを処理させる */
            if ((pid = fork()) < 0) {
                perror("fork");
                exit(1);
            }
            if (pid == 0) {
                /* サブプロセスの処理 */
                char *text = "send string: ¥"hello
client!¥"¥n";
                write(as, text, strlen(text));
                close(as);
                exit(0);
            } else
                close(as);
        }
    }
```


© 2011 NTT Service Integration Laboratories

４．プロトコルに依存する処理

© 2011 NTT Service Integration Laboratories

プロトコルごとに処理を分ける例

●アプリケーション層にIPアドレスに関する情報を埋め込む例

－FTP, SIP などが該当

■IPv4でのFTP(send-port)

```
USER ftp
PASS xxxx@example.com
PORT 192.168.0.21,14,248
RETR index.txt
```

■IPv6でのFTP(send-port)

```
USER ftp
PASS xxxx@example.com
EPRT |2|2001:db8:aaaa::21|3832|
RETR index.txt
```

●アプリケーションがIPアドレス情報を直接扱うケース

■Webサーバのログ生成・IPアクセスリスト処理など

```
192.168.0.21 - - [23/Oct/2011:18:11:29 +0900] "GET /index.html
HTTP/1.1" 200 2411 "-" "Mozilla/5.0 (Windows NT 6.1; WOW64)
AppleWebKit/535.1 (KHTML, like Gecko) Chrome/14.0.835.202 Safari/535.1"
2001:db8:aaaa::1 - - [23/Oct/2011:18:09:10 +0900] "GET /index.html
HTTP/1.1" 200 2411 "-" "Mozilla/5.0 (Windows NT 6.1; WOW64)
AppleWebKit/535.1 (KHTML, like Gecko) Chrome/14.0.835.202 Safari/535.1"
```


© 2011 NTT Service Integration Laboratories

プロトコル別の処理が必要になったら

```
struct sockaddr_storage ss;
struct sockaddr_in      *sin4;
struct sockaddr_in6     *sin6;

switch (ss.ss_family) {
case AF_INET:
   sin4 = (struct sockaddr_in *)&ss
    ...
   break;
case AF_INET4
   sin6 = (struct sockaddr_in6 *)&ss;
    ...
   break;
}
```


© 2011 NTT Service Integration Laboratories

NTT アプリケーション設定によるプロトコル指定

- ユーザが利用するプロトコルを強制できるようにする
  - クライアントプログラムの例
    - ssh -6：IPv6の利用を強制する：IPv4は利用しない
    - ssh -4：IPv4の利用を強制する：IPv6は利用しない
    - ssh　　：IPv6,IPv4両方の接続を試行する（デフォルト）
  - サーバプログラムの例
    - httpd -6：IPv6の利用を強制する：IPv4は待ち受けない
    - httpd -4：IPv4の利用を強制する：IPv6は待ち受けない
    - httpd　　：IPv6,IPv4両方の接続を待ち受ける（デフォルト）

■クライアントのコード

```
/* ポート22(ssh)にTCPで接続したい */
if ( "-4が指定された" )
    hints.ai_family = PF_INET;
else if ("-6が指定された")
    hints.ai_family = PF_INET6;
else
    hints.ai_family = PF_UNSPEC;

/*指定したアドレスファミリのリストだけが返却される */
g = getaddrinfo("www", "ssh",
                &hints, &ai0);
```

■サーバのコード

```
/* ポート80(http)のTCP接続を待ち受け */
if ( "-4が指定された" )
    hints.ai_family = PF_INET;
else if ("-6が指定された")
    hints.ai_family = PF_INET6;
else
    hints.ai_family = PF_UNSPEC;
hints.ai_flags = AI_PASSIVE;
/*指定したアドレスファミリのリストだけが返却される */
g = getaddrinfo(NULL, "http",
                &hints, &ai0);
```


© 2011 NTT Service Integration Laboratories

５．アドレス選択機構（RFC3484）によるアプリケーションの制御

© 2011 NTT Service Integration Laboratories

アドレス選択機構(RFC3484)によるアプリケーションの制御

- RFC3484の概要
  - ホストにおけるアドレス選択ルールを定義した仕様
- アドレス選択ルールとは

A. 宛先アドレスの選択基準

- ホストが**複数のあて先アドレス**を得た場合、どこに向けて通信を開始するか基準が必要

B. 送信元アドレスの選択基準

- ホストが**複数の送信元アドレス**の候補を持つ場合、通信を開始する時、ホスト自身がどれを使用するか基準が必要

アドレス選択の制御はスタックの設定のため上位アプリケーション(名前解決API)に共通的に作用。アプリケーションの改変も不要

※B送信元アドレスの選択については本チュートリアルでは割愛

© 2011 NTT Service Integration Laboratories

アプリケーション構造の視点からの IPv4/IPv6優先の制御

- サーバのホスト名 "www.example.com"を解決したところ、合計４つのアドレスが得られた場合の
  - IPv6 ①2001:db8:aaaa:80, ②2001:db8:bbbb::80
  - IPv4 ③192.16.178.80, ④192.16.179.80
- OS標準設定では getaddrinfo (3) の返却順序はIPv6が優先

①2001:db8:aaaa:80 → ②2001:db8:bbbb::80 → ③192.16.178.80 → ④192.16.179.80

- もしIPv4を優先して接続したい場合に、期待されるリスト構造は

③192.16.178.80 → ④192.16.179.80 → ①2001:db8:aaaa:80 → ②2001:db8:bbbb::80

ポリシデータベースを設定することでリスト構造を制御可能

© 2011 NTT Service Integration Laboratories

アドレス選択のためのポリシデータベース

優先度とラベルの値を書き換えることで、あて先アドレス(A)と送信元アドレス(B)の選択の優先順位を制御できる

Windows 7 のデフォルト設定のポリシデータベース

プレフィックス	優先度	ラベル	プレフィックスの意味
::1/128	50	0	ループバックアドレス
::/0	40	1	IPv6アドレス
2002::/16	30	2	6to4アドレス
::/96	20	3	互換アドレス
::ffff:0:0:/96	10	4	IPv4アドレス
2001::/32	5	5	Teredoアドレス

優先度 ↑ あて先アドレス選択に作用
優先度の高いものからマッチ

ラベル ↑ 主に送信元アドレス選択に作用

© 2011 NTT Service Integration Laboratories

NTT 優先度の設定によるあて先アドレスの制御

アプリケーションにIPv4を優先して接続させる設定例

プレフィックス	優先度	ラベル	プレフィックスの意味
::1/128	50	0	ループバックアドレス
::/0	40	1	IPv6アドレス
2002::/16	30	2	6to4アドレス
::/96	20	3	互換アドレス
::ffff:0:0:/96	60	4	IPv4アドレス
2001::/32	5	5	Tredoアドレス

IPv4アドレスの優先度（60）を、IPv6アドレスの優先度（40）より高く設定することで、名前解決API （getaddrinfo（3））はIPv4アドレスを優先したリストを生成する

③192.16.178.80 → ④192.16.179.80 → ①2001:db8:aaaa:80 → ②2001:db8:bbbb::80

© 2011 NTT Service Integration Laboratories

ポリシデータベースの実装状況

主要なOSでは、Windows XP/Vista/7, Linux 2.6.25以降, BSD系UNIX, Solaris 10以降に実装されている

■Windows 7 でのポリシデータベースの確認

```
C:¥>netsh interface ipv6 show prefixpolicies
アクティブ状態を照会しています...

優先順位      ラベル    プレフィックス
----------  -----  ------------------------
        50      0  ::1/128
        40      1  ::/0
        30      2  2002::/16
        20      3  ::/96
        10      4  ::ffff:0:0/96
         5      5  2001::/32
```

※MacOS X Lion (10.7) ではポリシーデータベースはread onlyで実装

© 2011 NTT Service Integration Laboratories

６. IPv6対応アプリケーションに関する最新動向

© 2011 NTT Service Integration Laboratories

IPv6→IPv4フォールバック問題

- デュアルスタックサーバへのアクセスで、IPv6通信に失敗した場合IPv4通信に切り替わりまで遅延時間が大きい事象がある
  - 例：IPv6ネットワークに障害が起きた場合にユーザは顕著に体感する可能性がある

デュアルスタックネットワーク

DNS登録

クライアント	
IPv6スタック	→ IPv6 →
IPv4スタック	

Webサーバ X	A と AAAA
IPv6スタック	
IPv4スタック	

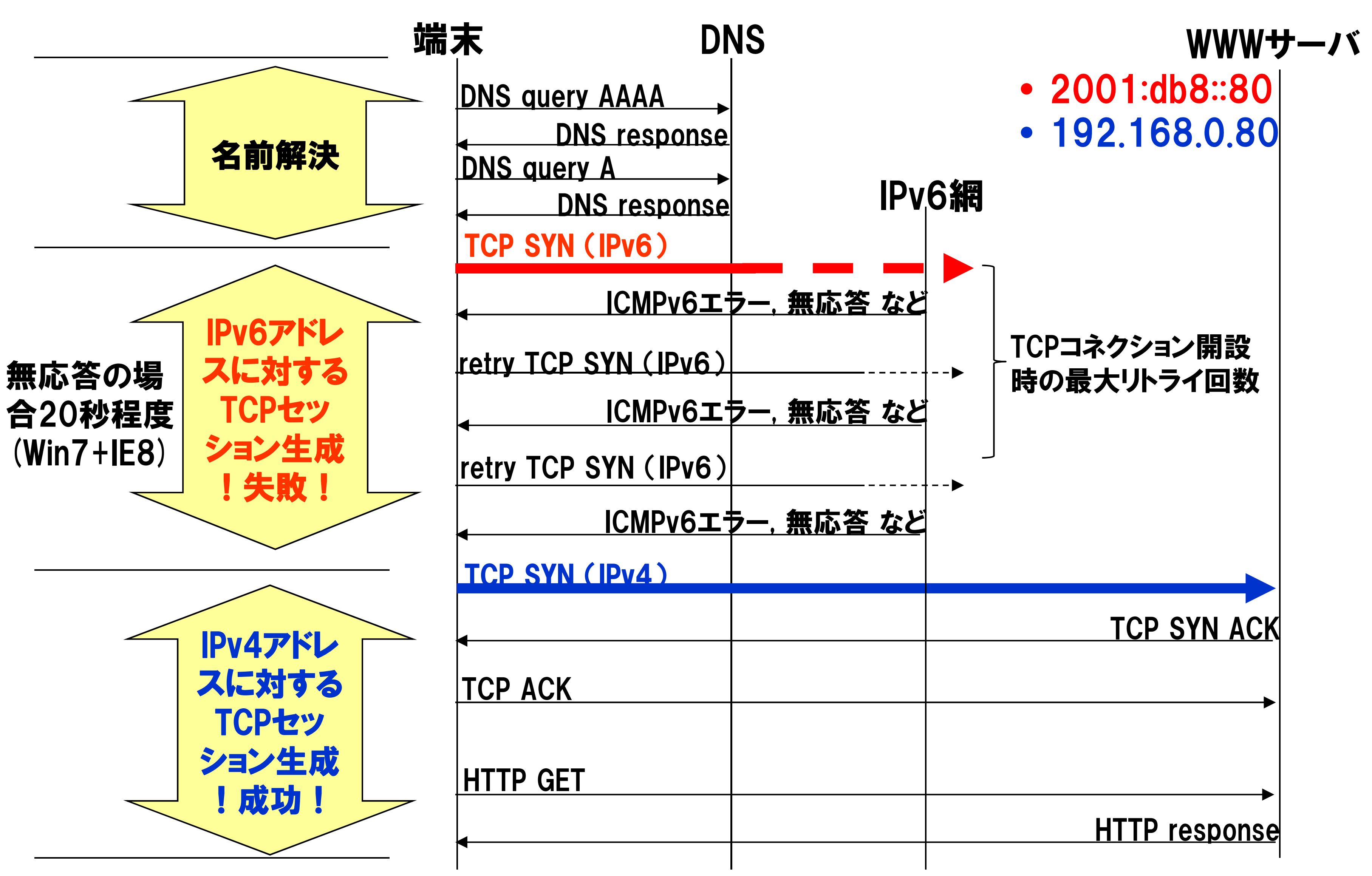
クライアント	
IPv6スタック	→ IPv6 → ×
IPv4スタック	→ IPv4 →

IPv6ネットワークに障害が発生

Webサーバ Y	A と AAAA
IPv6スタック	
IPv4スタック	

© 2011 NTT Service Integration Laboratories

IPv6→IPv4フォールバックのシーケンス例



© 2011 NTT Service Integration Laboratories

NTT

フォールバックによる遅延を回避するプログラム構造

Happy Eyeballs

- 複数のIPアドレスに対して、クライアントは同時にTCPセッションの開始を試み、最も速く確立できたIPアドレスと通信を行う方式

- Eyeballs：視聴者の意
  - Web閲覧者(視聴者)にとってはIPv4/IPv6は関係ない
  - IPv6ネットワークの障害や遅延に影響されないブラウジングを実現したい

- 標準化と実装状況
  - IETF v6ops WGで議論されている(*1)
  - 実装状況
    - Firefox 7以降, Google Chrome 14以降
    - MacOS X Lion以降のSafari, iPhone/iPad(iOS5)+Safari
    - Erlang

(*1) http://tools.ietf.org/html/draft-ietf-v6ops-happy-eyeballs

© 2011 NTT Service Integration Laboratories

Happy Eyeballs（cont.）

■Happy Eyeballsのシーケンス（基本パタン）

名前解決の結果
- IPv6アドレス1つ
- IPv4アドレス1つ

を得た

TCPコネクションのオープンを同時試行

Web#2が応答

端末はWeb#2（IPv4）との通信を選択

© 2011 NTT Service Integration Laboratories

Happy Eyeballs（cont.）

■Happy Eyeballsのシーケンス（Chromeでの実装例）

名前解決の結果
- IPv6アドレス1つ
- IPv4アドレス1つ

を得た

IPv6接続を優先させつつ、IPv6ネットワークやサーバでの遅延や障害を回避できる

© 2011 NTT Service Integration Laboratories

7. まとめ

© 2011 NTT Service Integration Laboratories

本チュートリアルのまとめ

- アプリケーション設計の基本方針
  - RFC3493が参照するPOSIX API (getaddrinfo(3))を活用し、プロトコルに依存する記述を極力排除する
- IPv4 onlyのアプリケーションでも新しいAPIは有効
  - 名前解決近辺のコードが非常に簡潔に記述できる
  - IPv4を強制するフラグもあるので、IPv6の予期せぬ副作用を抑止することも可
- RFC3484 ポリシーテーブルの設定により、アプリケーションを改変することなく優先プロトコルを制御可能
- Happy Eyeballs のような新しい仕組みがアプリケーション設計に取り入れられつつある

© 2011 NTT Service Integration Laboratories

参考文献

1. "Basic Socket Interface Extensions for IPv6", R. Gilligan, S. Thomson, J. Bound, J. McCann, W. Stevens, Feb. 2003, RFC 3493.
2. "Advanced Sockets Application Program Interface (API) for IPv6", W. Stevens, M. Thomas, E. Nordmark, T. Jinmei, May 2003, RFC 3542.
3. 「IPv6ネットワークプログラミング」, 萩野純一郎 著、小川彩子 訳, 2003年2月, アスキー.
4. "Unix Network Programming Volume 1., 3rd Edition, The Sockets Networking API", W. Richard Stevens, Bill Fenner, Andrew M. Rudoff, Nov. 2003, Prentice Hall.
5. "Default Address Selection for Internet Protocol version 6 (IPv6)", R. Draves , Feb. 2003, RFC 3484.
6. "Happy Eyeballs: Success with Dual-Stack Hosts," D. Wing, A. Yourtchenko, Sep. 2011, draft-ietf-v6ops-happy-eyeballs.

© 2011 NTT Service Integration Laboratories