アプリケーションの IPv6対応概要

株式会社リコー
研究開発本部
基盤技術開発センター
大平浩貴 (おおひら こうき)

IPv6の進展

RICOH
imagine. change.

- ISP→続々対応中
- iDC/ホスティング→続々対応中
- 端末OS→対応済み（随時機能向上中）

- …じゃぁ、アプリは？
  - ApacheやBINDなど、公共性の高いものは対応済み

- …じゃぁ、私たちが作るアプリは？
  - 私たち自身がこれからがんばらないと！

IPv6プログラミングの情報について

RICOH
imagine. change.

- 今回の解説で参考にしている書籍
  - IPv6ネットワークプログラミング ASCII社刊
    - http://ascii.asciimw.jp/books/books/detail/4-7561-4236-2.shtml
  - 著者は萩野潤一郎 (itojun) 氏
  - 今回参考にしたプログラムもこの本に掲載されているもの
    - itojun氏が製作し、パブリックドメインで公開
- IW2011のT5「IPv4アドレス枯渇時代のアプリケーション開発」セッション
  - http://www.nic.ad.jp/ja/materials/iw/2011/proceedings/t5/

IPv6普及･高度化推進協議会での活動

RICOH imagine. change.

- IPv6/IPv4共存WG アプリケーションIPv6化検討SWG
  - http://www.v6pc.jp/jp/wg/coexistenceWG/v6app-swg.phtml

- WebアプリのIPv6化
  - 現在調査･検討中
  - 公開を目指して活動中
- socketベースアプリのIPv6化
  - パブリックコメント受付完了
  - 第1版公開予定

今回の説明の概要

RICOH
imagine. change.

- BSD Socket APIを使用したアプリケーションソフトウェアのIPv6化を説明
- クライアントプログラムのIPv6対応
  - 実際の手順
- サーバプログラムのIPv6対応
  - 手法の分類
  - 実際の手順
- 名前解決の問題と解決案
- 組み込みの話

RICOH
imagine. change.

BSD Socket による クライアントアプリケーション

デュアルスタック対応とは

RICOH
imagine. change.

- IPv4対応プログラム（シングルスタック）
  - ひとつのプロトコルに対応していた
- IPv6/IPv4両対応プログラム（デュアルスタック）
  - 複数のプロトコル・複数アドレスの中のどれで送信を行うか選ばなければならない
  - 複数のプロトコル・アドレスで同時に受信しなければならない

ただ単に関数を変更するだけではだめ
選択機構や、並列受信機構などが新たに必要となる

従来のクライアントプログラミング

RICOH
imagine. change.

- 大まかな流れ
  - ホスト名解決
  - サービス名解決
  - Socket生成 (ファイルデスクリプタ生成)
  - Connect実行 (通信相手指定・接続を確立)
  - デスクリプタによる入出力
  - クローズ
- 以降ではシングルスタック環境の関数と、それを置き換えるデュアルスタック環境の関数を挙げる

ホスト名・サービス名解決

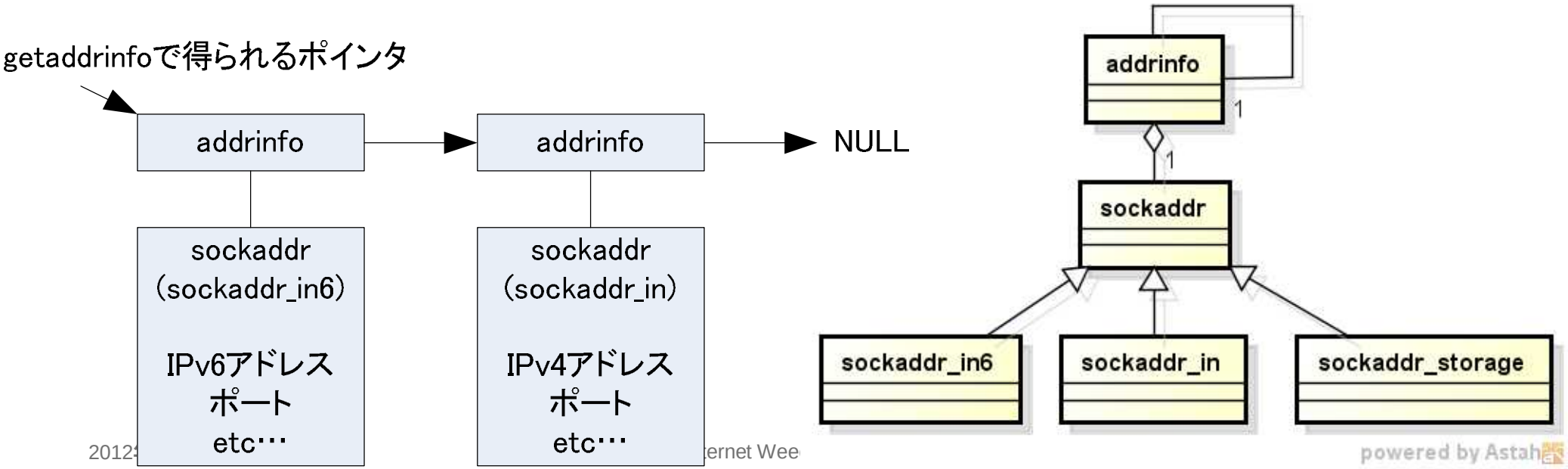
RICOH
imagine. change.

- IPv4
  - ホスト名：*gethostbyname()*でhostent構造体を得る
  - サービス： *getservbyname()*でservent構造体を得る
- デュアルスタック
  - *getaddrinfo()*でaddrinfo構造体のリストを得る
    - リストの開放も可能で、*freeaddrinfo()* 関数による
- 注意
  - *gethostbyname2()* はIPv6を扱えるが使うべきではない

addrinfo構造体とsockaddr構造体

RICOH
imagine. change.

- addrinfo構造体
  - 複数のアドレス情報をLinked Listで保持する
  - 内部でアドレスを保持するsockaddr構造体へのリンクを持つ
- sockaddr構造体
  - 各種アドレス情報を汎化した構造体
  - 実体はsockaddr_in6 (v6) やsockaddr_in (v4)
  - どちらが入るかわからない場合はsockaddr_storageで定義



逆引き

RICOH
imagine. change.

- IPv4
  - アドレス：*gethostbyaddr()*でhostent型構造体を得る
  - サービス：*getservbyport()*でservent構造体を得る
- デュアルスタック
  - *getnameinfo()*とaddrinfo構造体からホスト名やサービス名の文字列を取得できる
  - いろいろなオプションが指定可能
    - NI_NOFQDN …FQDNではなくホスト名だけ
    - NI_DGRAM …UDPのポート情報を得る
    - etc…

IPアドレスの内部表現と文字列の変換

RICOH
imagine. change.

- IPv4
  - *inet_ntoa()* でin_addr型を文字列へ変換
  - in_addr型はIPv4限定
- デュアルスタック
  - *getnameinfo()*とaddrinfo構造体からIPアドレスの文字列表現が得られる
  - 本来は逆引き関数だが、NI_NUMERICHOSTオプションを指定することで、ホスト名ではなくアドレスの文字列表現が得られる

ソケット生成･コネクト

RICOH
imagine. change.

- デュアルスタックの場合addrinfo構造体を参照する
  - 例：addrinfo ai;
  - プロトコルファミリ： ai->ai_family
  - ソケットタイプ： ai->ai_socktype
  - プロトコル： ai->ai_protocol
  - アドレス： ai->ai_addr
  - アドレス長： ai->ai_addrlen
- 実例

```
s=socket (ai->ai_family, ai->ai_socktype, ai->ai_protocol) ;
connect(s, ai->ai_addr, ai->ai_addrlen);
```

クライアントのコード概要

RICOH
imagine. change.

```
struct addrinfo hints, *res, *resall;
hints.ai_family = AF_UNSPEC;
hints.ai_socktype = SOCK_STREAM;
getaddrinfo(“www.v6pc.jp”,”http”, &hints, &resall);

for (res = resall; res; res = res->ai_next) {
        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){
                close(s);
                continue;
        }
        /*読み書き*/
        close(s);
        break;
}
```

デュアルスタッククライアントのまとめ

RICOH
imagine. change.

- *getaddrinfo()*でアドレスのリストを得る
  - addrinfo構造体のリスト
- リストの順にソケット生成・接続を行い、成功したら通信して終了する

RICOH
imagine. change.

BSD Socket による サーバアプリケーション

サーバのデュアルスタック化

RICOH
imagine. change.

- いくつか手法がある
  - inetdを使用する
  - 自身のプロトコル・アドレスの数だけ*socket()*を生成して、全てのFDに対して応答処理をする
  - シングルスタック仕様のプログラムを複数プロセス走行させる
  - IPv4 マップドアドレス (IPv4 Mapped IPv6 address) を使用して、v6ソケットで通信する

サーバアプリ実現手法の分類

RICOH
imagine. change.

手法	利点	欠点
【手法1】 inetdを使用する	通信部分をinetdが代行するため、通信のIPv6化を意識しなくてよい	inetdを必要とする
【手法2】 複数のsocketを生成する	ひとつのプロセスでマルチプロトコルに対応できる	複数ソケットを生成し、それらを同時に待つため、プログラムが複雑になる
【手法3】 シングルスタックプログラムを複数プロセス走行させる	プログラム構成の変更なしにIPv6に対応できる	共有リソースを扱う場合、プロセス間で排他制御する必要がある
【手法4】 IPv4マップドアドレスを使用する	ひとつのソケットでIPv4/IPv6両方を処理でき、プログラム構成の変更が必要ない	IPv4とIPv6の処理が混在する。アドレスを扱う際にはIPv4マップドアドレスかどうかの判定が必要となる場合もある

inetdによるデュアルスタックサーバ

RICOH imagine. change.

手法1

- 通信部分は変わらない
- 通信相手アドレスを取得する部分で注意が必要
  - *getpeername()* FDとsockaddr構造体を引数で渡すとsockaddr構造体に相手ピアアドレスを書く
  - 引数は sockaddr構造体ではなく、sockaddr_storage構造体を使用する
    - sockaddr_storage構造体はどんなプロトコルのアドレスでも記憶できる領域を持つ

```
sockaddr_storage from;
getpeername(0,(sockaddr*)&from,sizeof(from));
```

  - あとは*getnameinfo()*で文字列化

複数socketで待ち受けるサーバ

RICOH
imagine. change.

手法2

- もっとも典型的で理想的な対応
- プログラム構成が変化する
  - 複数のデスクリプタを同時待ち受けする機構
- 完全な新規で設計する通信プログラムはこの構成が望ましい

シングルスタックサーバアプリの流れ

RICOH
imagine. change.

- ソケット生成
- *bind()*でソケットにアドレス・ポートを割り当てる
- *listen()*で接続待ちソケットとしてキューを割り付ける
- 以降はループ
  - *accept()*で接続待機
  - *read()* | *write()*で通信
  - *close()*で終了・ループ先頭へ

複数ソケットを処理するサーバの流れ

RICOH
imagine. change.

- *getaddrinfo()* で自身のプロトコル・アドレスを *addrinfo* 構造体のリストで得る
- リストで得られたプロトコル・アドレス個別に下記を実施
  - *socket()*、*bind()*、*listen()*
- 得られた複数のFDをfd_set構造体に保存
- 以降はループ
  - fd_set構造体を引数に *select()* で接続の待機
  - *select()* を抜けてきたfdに対して *accept()*
  - 読み書き処理
  - クローズ

自身のプロトコル・アドレスリストを得る

RICOH
imagine. change.

- 自身のプロトコル・アドレス一覧を得る

```
addrinfo hints,resall;
hints.ai_socktype = SOCK_STREAM;
hints.ai_flags = AI_PASSIVE;
hints.ai_family = AI_UNSPEC;
getaddrinfo ( NULL,”http”, &hints, &resall);
```

- resallにプロトコル・アドレスのリストが保存される

自身のアドレスリストのループ

RICOH
imagine. change.

- ループでsocket生成～listenまで行う。

```
for( res=resall ; res ; res=res->ai_next ){
        s[i] = socket(res->ai_family, res->ai_socktype,
                                        res->ai_protocol);
        bind( s[i], res->ai_addr, res->ai_addrlen);
        listen( s[i], 5);
        i++;
}
```

得られたFDをfd_setへ

RICOH
imagine. change.

- fd_setを作成する
  - fd_setとは、selectで待ち合わせするFDをまとめた構造体を指す

```
fd_set rfd0;
for( i=0; i< (listen成功したソケット数) ; i++)
        FD_SET(s[i], &rfd0);
```

- ついでにFD（すなわちs[i]）の最大値を求めておく
  - 後のselectのために

selectで待ち合わせて読み書き

RICOH
imagine. change.

接続受付･送受信のループをまわす

```
for(;;){
        rfd = rfd0;
        select( (FD最大値+1) , &rfd, NULL, NULL, NULL);
        for(i=0 ; i<(ソケット最大数) ; i++){
                if(FD_ISSET(s[i],&rfd)) {
                        accept(s[i], &from, &fromlen);
                        /*読み書き*/
                        close(s[i]);
                }
        }
}
```

複数プロセスで待ち受けるサーバ

RICOH
imagine. change.

手法3

- シングルスタックアプリを複数走行させる
  - *getaddrinfo()* で AF_INETとAF_INET6のどちらかを設定する
- forkでひとつのプログラムがv4/v6に分離するようにすればリソースの節約も可能
  - Copy on Write機能による
- bind時のINADDR_ANY (IPv4) に相当するのがIN6ADDR_ANY_INIT (IPv6) となる

デュアルスタックサーバのまとめ

RICOH
imagine. change.

- いくつか手法があるので、メリットとコスト・リスクを比較して適切な選択をしましょう

RICOH
imagine. change.

名前解決の問題と最近のテクニック

getaddrinfoの並びは？

RICOH
imagine. change.

- *getaddrinfo()* は名前解決
  - 出力されるaddrinfo構造体のリストはどういう順序になるのか？
- 長らく RFC3484で定義されていた
- 2012年9月に RFC6724 がRFC3484をObsoleteした
  - デフォルトポリシーテーブルの修正
  - アドレス選択ルールの修正
  - フォールバック問題の記述
  - etc...

フォールバック問題

RICOH
imagine. change.

- フォールバックとは
  - クライアントが接続先IPアドレスのリストを得る
  - リストの先頭にあるIPアドレスに接続しようとして、失敗すると次のリスト要素のIPアドレスを試す
- 原因
  - サーバがそのプロトコル･IPアドレスでアプリサービスをしていない
  - ネットワークの接続性が失われている
- 問題
  - タイムアウトを繰り返すので、接続まで時間がかかる
  - 環境によっては数十秒かかる場合も見込まれている

フォールバック問題への対処

RICOH
imagine. change.

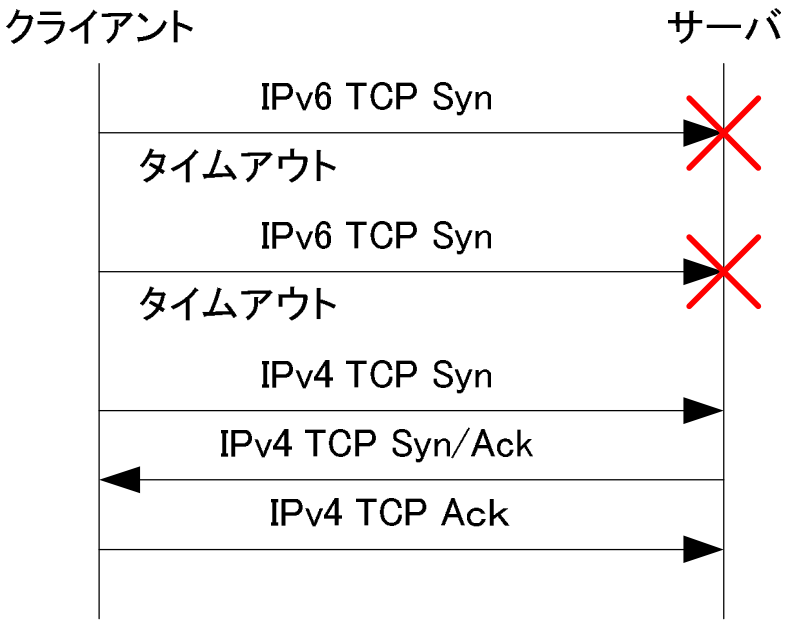
- サーバがサービスしていないIPアドレスはDNSに登録しない
- IPの接続性を健全に保つ
- ポリシーテーブルを変更する
  - Windows: netsh interface ipv6 show prefixpolicies
  - Linux: ip addrlavel show
  - FreeSBD: ip6addrctl show
- サーバサイドだけでの対応では完全には解決できない

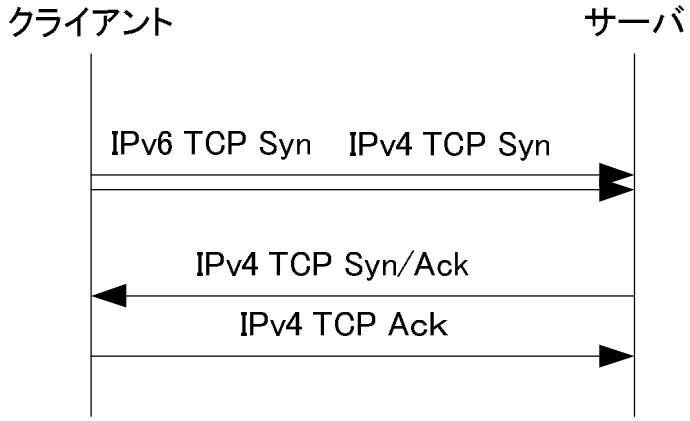
Happy Eyeballs

RICOH
imagine. change.

- RFC6555、RFC6556で定義
- 従来はTCP Synの接続失敗を受けて次のTCP Synを送っていた

- Happy EyeballsはIPv6 / IPv4アドレスに両方に対して一気にTCP Synを送る
  - Syn/Ack応答が帰ってきたIPにTCP Ackを送って接続
- 実装依存性が非常に高い
  - 複数I/Fの場合やIPv6アドレスが複数ある場合は？
  - IPv6 Syn/Ackが遅れて届いたら？
  - 今後の展開や運用を注視すべき





組み込み用途でのIPv6対応

組み込み機器へのアドレス埋め込み

RICOH
imagine. change.

- 組み込みではSocketを使うことが多い

- 組み込み機器はいろいろ大変
  - お客様の環境でDNSが使えない
  - 名前解決処理に必要なリソースが不足している
- 組み込み機器でIPアドレスをハードコーディングしたいこともある
  - しかしやめたほうがいい
  - RFC4085でこの問題が記述されている

アドレスハードコーディングの問題

RICOH imagine. change.

- そもそもIPアドレスは借り物
  - リナンバリングのリスクが伴う
- ホスト名は名前指定して、名前解決には*getaddrinfo()*を使うべき
  - RFC6724やRFC3484 に従った適切な処理が約束されている
  - これを自作するということはRFCに従うコストやそれから外れるリスクを自己負担するということ
- これらのリスク・コスト・インターネットの道義的責任がハードコーディングしないリスク・コストを上回ったとき
  - 十分な注意・サポートとともにやむをえずハードコーディングすることは考えられる

最後に

RICOH
imagine. change.

- IPv6はどんどん浸透してきている
  - アプリでIPv6を先取りして、時代もお客様も先取りしよう
- 何かありましたらいつでもこちらまで
  - IPv6普及･高度化推進協議会の連絡先
    - v6info@v6pc.jp
    - https://www.v6pc.jp/jp/info/inquiry_web.phtml
  - 大平の連絡先
    - ohhira@src.ricoh.co.jp
    - kohki@lemegeton.org
    - Twitter：@torawarenoaya
    - facebook：http://www.facebook.com/kohki.ohhira
    - LinkedIn：http://jp.linkedin.com/pub/kohki-ohhira/10/7ba/6a2