アプリケーションの IPv6 対応ガイドライン
Socket 編

2012 年 5 月 1 日

IPv4/IPv6 共存 WG
アプリケーションの IPv6 対応検討 SWG

目次

変更履歴

版	改版日	摘要
0.9	2012 年 5 月 1 日	パブリックコメント版

1 本書について

2011 年の IPv4 アドレス在庫枯渇を機に、通信事業者や ISP 各社から IPv6 対応のサービス展開が進み、インターネットのユーザフロント、サーバフロント共にこれまでの IPv4 での運用から IPv6 が混在する IPv4 と IPv6 の共存期に入りつつあります。一方で、スマートフォンに代表されるスマートメディアが普及し、ネットワークを使うアプリケーションは一般的になっています。こうした状況の中、アプリケーションの開発者が目にする情報は、これまでの IPv4 で運営されているネットワークを前提としたものが多く、共存状況で注意すべきことやするべきことが整理されていませんでした。

IPv6 普及・高度化推進協議会 IPv4/IPv6 共存 WG では、2011 年 9 月に「アプリケーションの IPv6 対応検討 SWG」を発足し、共存期を前提としたアプリケーション開発についての情報整理を行い、アプリケーション開発者に向けた情報発信と情報共有の活動を始めました。

本書は、この「アプリケーションの IPv6 対応検討 SWG」の活動に基づいたソケットを用いたプログラム手法の IPv6 対応方法についてのまとめ資料です。最近は、直接ソケットを用いたプログラムを行わずにクラスライブラリなどを利用し簡単にネットワークを利用できる言語も多くなっていますが、基礎的な部分でどのような処理やデータのやり取りが行われ、その際 IPv4 と IPv6 では何が違うのかを本書を通じてご確認頂けば幸いです。

2 BSD ソケットによるプログラミングの流れ

通信プログラムを実装するために BSD ソケットやそれに類似したソケット関数群を用意している処理系は多い。
BSD ソケットは BSD 系 UNIX を起源としており、もともとは C 言語の API である。
基本的に下記の関数を順に呼び出すことで、サーバでのクエリ待ちうけやクライアントからの接続を実現するものである。

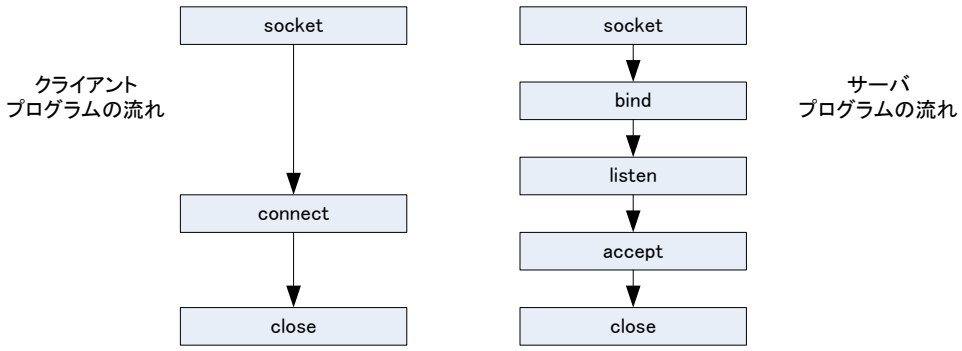

図 1　　ソケット API 呼び出しの基本的な流れ

アプリケーションを IPv6 に対応させる場合でも、上記のソケット関連関数の役割や呼び出し順序は変わらない。
しかし、IPv4 のみに対応したプログラミングと IPv4/IPv6 に対応したプログラムでは下記の部分が異なる。

- サーバが待ち受ける IP アドレスやクライアントが接続する先の IP アドレスを得る手段
- IP アドレスからホスト名を得る手段
- IP アドレスを文字列表現するなどの手段
- 待ち受ける処理を複数プロトコル・複数アドレスで並行処理する

3 IPv6 対応ソケットプログラミング

3.1 基本的方針

特別な環境を除いて一般的に、従来から IPv4 で提供されてきたサービスは、今後もしばらく IPv4 でサービスし続けることが求められる。このため、そのようなアプリケーションを IPv6 に対応させるとしても、IPv4 にも継続して対応しなければならない。
このため、既存アプリケーションの IPv6 対応とは、IPv4 / IPv6 両方、すなわちマルチプロトコルに対応することになる。
クライアントにおいては、従来は名前解決の結果得られた単一アドレスに接続し、そこにクエリを送信して応答を得るものであった。これが IPv6 対応した場合、名前解決で複数のプロトコルの複数のアドレスが得られることになる。この複数のアドレスに対して順に接続して通信可能なものを選ぶという処理になる。
サーバにおいては、従来は単一のプロトコル（IPv4）を監視し、そこに到着したクエリに

応答するだけであった。これが IPv6 対応となった場合、複数のプロトコルを監視し、そのいずれかに到着したクエリに応答を返すことになる。
特にサーバプログラミングは下記のような手法が考えられる。

表 1　　サーバプログラミングの手法と特徴

手法	利点	欠点
複数のsocketをオープンするプログラムにする。	ひとつのプロセスでマルチプロトコルに対応できる。応用しやすい。	複数socketを生成し、それらをselectで待つため、多少プログラムが複雑になる
プログラムでIPv4 listenモードとIPv6 listenモードの二つを作り2つのプロセスを走行させる。	比較的簡単にIPv6に対応できる。	複数プロセスが走行するためのリソースが必要。（ただしCopy on Write や Demand Pagingのため、VM消費は抑制される。） 競合するリソースを扱う場合、プロセス間で
inetdのtcpないしはtcp6から呼び出し可能なサーバにする。	簡単に実現できる。	inetdを必要とする。
IPv4 Mapped IPv6 Addressを使用する	ひとつのソケットでIPv4/IPv6両方を処理できる	場合によってはIPv4とIPv6の処理が混在することになる。アドレスのUIなどではMapped Addressかどうかの判定とそれに応じた処理が必要となるケー

3.2 基本的な RFC

現在、BSD ソケットを IPv6 に対応させる API を定義した RFC は 2 系統ある。
一方は基本的なソケット API の IPv6 化を定めた文書であり、最新は RFC3493 である。
もう一方は詳細かつ高度なソケット API の IPv6 化を定めた文書であり、最新は RFC3542 である。

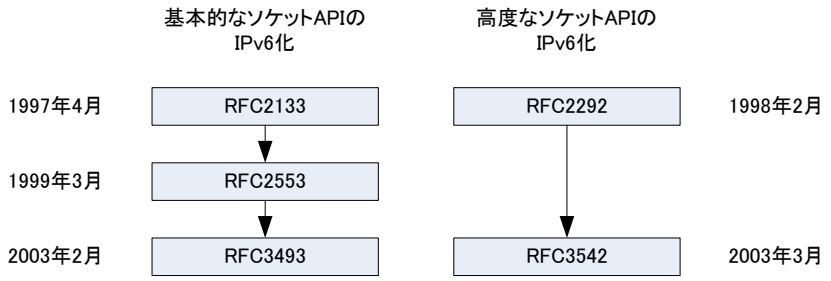

図 2　　RFC 改訂の流れ

前者の RFC はプロトコルファミリの追加、IPv6 アドレス構造体の定義、ソケット関数のシンタックスには変更のないことと IPv4 との互換性、各種アドレス表記、インターフェース ID などの基本的事項について記述している。
後者の RFC は、IPv6 ヘッダの詳細、icmpv6 ヘッダの詳細、Raw ソケット、IPv6 拡張ヘッダ、ソケット付随情報の拡張、パケットの情報（Source/Destination アドレスや Hop Limit など）、ルーティングヘッダオプション、その他のオプション、MTU 関連機能、IPv4 mapped IPv6 アドレスなどについて記載されている。

3.3 書籍

下記の書籍が既存アプリの IPv6 化を含めて、基本的なところから詳しく説明している。
IPv6 ネットワークプログラミング 荻野純一郎（itojun）氏著 ISBN4-7561-4236-2

3.4 RFC4038 における記述

RFC4038 ではアプリケーションプログラムを IPv6 に対応させる際の問題、移行シナリオ、アプリケーションの各レイヤそれぞれのポーティングに関する検討、IP のバージョンに依存しないアプリケーションの作り方について述べている。

4 ソケットプログラミングの実際

4.1 パブリックドメインソフトウェアによる解説

前期の書籍「IPv6 ネットワークプログラミング」にも掲載されているが、荻野純一郎（itojun）氏がサンプル実装をパブリックドメインソフトウェアとして公開している。
その情報を当該資料に添付する。なお、itojun 氏の記述したソースにあった、現在の C 言語記法に照らし合わせて古い記述方式であった main 文のプロトタイプ宣言部分は削除している。

4.2 IPv6 対応クライアントプログラミング

従来は gethostbyname 関数や getservbyname 関数を使用して接続先ホストの名前解決を行っていた。IPv6 対応クライアントでは、これらの関数を使用せず、getaddrinfo 関数を

使い、addrinfo 構造体のリストの形で相手先ホストの IP アドレス列を取得する。
また、従来はひとつのアドレスに接続してクエリを送信・応答を取得していた。IPv4 および IPv6 の両方に対応するためにはこの構造を改め、addrinfo 構造体のリストを順に辿りつつ接続を繰り返し、接続に成功したらクエリを送信・応答を取得する構造にする。
この様子を添付している itojun 氏によるサンプルプログラムで示している。

4.3 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

従来は gethostbyname 関数や、getservbyname 関数などが使用されていた。IPv6 対応プログラミングではそれらの関数の使用をやめ、IPv4/IPv6 双方を汎化した addrinfo 構造体や、それを使用する getaddrinfo 関数や getnameinfo 関数を使用する。

getaddrinfo 関数を使うことで、自身の listen 可能なアドレスすべてをリストの形で得られる。このアドレス一つ一つについて socket を生成し、bind、listen する。
得られた複数のデスクリプタについて、select で待ち合わせる。
それらのソケットに接続が到着次第、該当ソケットに accept してクエリを読み込み、応答を返す。
この様子を添付している itojun 氏によるサンプルプログラムで示している。

4.4 マルチプロセスによる IPv6 対応サーバプログラミング

起動時の引数で、IPv6 による listen を行うか、IPv4 による listen を行うかを排他的に選択する。なお、その際に自身の IP アドレスを得るために getaddrinfo 関数を使用する。
複数アドレスでは待ち合わせをしないため、プログラムの構造は従来の IPv4 のみに対応したプログラミングとあまり変わらず、そのまま socket 生成、bind、listen、accept を行う。ただし、このプロセスが平行して複数走行することになるため、プロセス間でひとつのリソースを競合しないようにアクセスする機構が別途必要となる。
この様子を添付している itojun 氏によるサンプルプログラムで示している。

4.5 inetd を使用した IPv6 対応サーバプログラミング

inetd を使用することにより、サーバプログラムの通信処理は標準入出力に対する読み書きとして実装できる。このため、IPv6 に対応してもプログラムの主たる流れには影響がない。ただし、ロギングなどの用途で IP アドレスを用いる部分には修正が必要となる。

具体的には、getpeername 関数で接続相手を得ることができ、そこで得られた sockaddr 構造体を NI_NUMERICHOST 引数とともに getnameinfo 関数を呼び出して相手ホストの文字表記 IP アドレスを得る手順となる。文字表記の IP アドレスを得るために従来の ntop 関数は使用すべきではない。
この様子を添付している itojun 氏によるサンプルプログラムで示している。

5 名前解決についての議論

IPv6 アプリケーションを構築するに当たっては、名前解決も重要な問題となる。
デュアルスタック環境でプログラムが正しく動作するには、アプリケーションがデュアルスタックに対応しているかどうかだけでなく、名前解決の結果が正しくアプリケーションに与えられなければならないためである。

5.1 RFC3484 の記述とその実現

IPv6 環境は、自身のインターフェースにも、相手ホストのインターフェースにも複数の IP アドレスが割り振られることになる。
そのため、自身から相手ホストに対して接続する場合、複数ある自分のアドレスのうちどれをソースアドレスとしてパケットに記載し、複数ある相手アドレスのどのアドレスをディスティネーションアドレスとしてパケットに記載するかが問題となる。
RFC3484 はホストが IP で接続する際に、ソースアドレスとディスティネーションアドレスをどのように選択するべきかを記載している。
IPv6 対応ソケットプログラミングにおける名前解決は前述のとおり主に getaddrinfo 関数で行う。具体的には getaddrinfo で名前解決を行うと候補となる IP アドレスをリストの形でユーザに返す。この getaddrinfo 関数が返すアドレスリストは RFC3484 に従った順序でソートされることとなっている。
すなわち、正しく実装された OS・言語環境の getaddrinfo 関数を使用する限りにおいては、RFC3484 に従った名前解決が実現されるということになる。
RFC3484 ではポリシーテーブルと呼ばれるアドレス候補順序ルールリストと、それに付随した名前解決順序のアルゴリズムが記述されている。
同 RFC の 2 章でポリシーテーブルの機能とデフォルトポリシーテーブルが記述されており、優先順位として、IPv6 ネイティブアドレス、6to4 アドレス、IPv4 互換アドレスと並んでいる。またこのテーブルが管理者によって書き換えられることにより、特殊なネットワーク環境にも対応できる。

また、ソースアドレス選択ルールについては同 RFC の 5 章で 8 つのルールが記述されており、ディスティネーションアドレスの選択ルールについては同 RFC の 6 章で 10 のルールが記述されている。
前述のとおり、getaddrinfo の結果はこのポリシーテーブルとルールに従った順序で返答されることとなる。

5.2 フォールバックの発生

ホスト名を名前解決した場合、RFC3484 では IPv4 よりも IPv6 のアドレスを優先的に返すポリシーとなっている。
しかし、サーバのインターフェースが IPv6 に対応おり、かつ DNS で A レコードだけでなく AAAA レコードが引けたとしても、サーバアプリケーションが IPv6 に対応していなければ、クライアントアプリは IPv6 による接続で失敗してしまう。またサーバアプリが IPv6 に対応異していた場合でも、途中経路が IPv6 に対応していなければ、やはりクライアントアプリは IPv6 の接続で失敗してしまう。
このような問題は IPv4 から IPv6 の移行期に発生する可能性があると見込まれている。
この場合、getaddrinfo が返す「IPv6 が優先的に配置されたリスト」を先頭から順に接続しては失敗を繰り返し、IPv4 アドレスがリストに現れて初めて接続されるということが起きる。この接続失敗から再試行の手順をフォールバックと呼ぶ。フォールバックが行われるトリガは TCP Syn 接続への応答がタイムアウトすることであり、実際に IPv4 で接続されるようになるまでにタイムアウトによって相当な時間が（場合によっては数十秒以上）かかってしまうことも見込まれている。
フォールバックを発生させないためには、IPv6 でサービスを行っていないサーバでは IPv6 アドレスを AAAA レコードに登録しないなどの対策が必要となる。

5.3 フォールバック挙動を最適化する Happy Eyeballs 提案

この問題をクライアントサイドで解決するために 2010 年末頃より IETF で提案されたのが Happy-Eyeballs という通信手順である。
RFC6555　Happy Eyeballs: Success with Dual-Stack Hosts で定義されている仕様であり、また RFC6556　Testing Eyeball Happiness でその動作検証方法が定義されている。
この技術は名前解決の後に TCP で通信するときの動作に関する提案である。
通常の IPv6 プログラミングでは名前解決後、IPv6 によって接続するために まず IPv6 で TCP Syn パケットを送信するが、この internet-draft ではその応答を待たず、直後に IPv4

でも TCP Syn パケットを送信することを提案している。このように IPv6 だけでなく IPv4 でも TCP Syn を送信し、TCP Syn,Ack が応答されたプロトコルに対して TCP Ack を再度送って TCP コネクションを張るという通信方式である。
これにより TCP コネクションが張られるまで数十秒がかかっていたものが数百ミリ秒で接続できるようになったという実験結果も挙がっている。IPv4 から IPv6 への移行期において、十分な IPv6 接続が望めない環境ではこの Happy-Eyeballs のギミックは有効に動作すると考えられる。
しかしながら、Happy-Eyeballs 提案には下記の問題がある。

- RFC3484 の定義とは合致しない。
- Happy Eyeballs は実装依存性が高い。IPv4 Syn を送信するタイミングは、IPv6 Syn 送信後「Immediately」とだけ書かれており、複数の Syn 送信とそれに対する Syn,Ack の具体的なタイミングが不明瞭である。
- キャッシュの構成や挙動に応じた動作について検討が十分とはいい難い。

以上のような問題があるため、Happy Eyeballs 手順を実装する場合は十分な考慮と動作実験が必要である。
いずれにせよ、Happy Eyeballs については今後の標準化や実装をウォッチしていく必要がある。

5.4 組み込み環境でのホスト名利用

組み込みシステムも PC と同様に他の機器と通信する機能を持つことが多く、その動作環境では DNS が利用可能ではないケースも十分にありえる。このような環境に対応するため、従来、接続先ホストを特定するために内部 ROM に接続先 IPv4 アドレスを直接記憶することもあった
しかし、IPv6 環境においては、IPv6 アドレスを直接 ROM 内に記憶することは避けるべきである。この理由は下記の 2 点による。

第一に IPv6 アドレスはユーザに貸し付けられるものであり、リナンバリングの可能性がある。IPv6 アドレスを組み込み機器の ROM 内に記載すると、接続先アドレスがリナンバリングされた場合に対応することができず、許可されていないアドレスを不当に使用したり、それにより動作不良を引き起こすことになる。このことから IPv6 アドレスの指定は DNS のような外部データベースを使用するべきである。この IP アドレスを組み込み機器の内部にハードコーディングする問題と、IP アドレスのハードコーディングを回避する手段について RFC4085 で言及されている。

第二に稼働環境でDNSなどのようなIPアドレスに依存しないサービスが利用できないケースにおいて、アドレスの公共性やリナンバリングなどのリスクを受容する場合であっても、IPv6 アドレスを直接使用せず、ホスト名の名前解決を hosts（OS 管理下のホスト名管理データベース）に任せ、プログラム側では getaddrinfo などの標準的関数が利用できるようにすべきである。これは前述の RFC3484 でアドレス選択順序が定義されており、それが正しく実装された OS 環境を利用することで、適切な名前解決・IP アドレス選択ルーチンを利用できるためである。それを利用しない場合、アドレス選択をメーカーのコード自身が行うことになる。これは、RFC などで検討された方針に従うコストや、それを逸脱するリスクをメーカー自身が負うことになる。
前記 2 点の上記のコスト・リスクが IP アドレスを直接扱わないリスクを下回った場合にのみ IP アドレスの公共性に十分に配慮して IP アドレスを直接扱うものとして、通常はDNSによる名前解決やそれを利用するgetaddrinfoなどの標準関数を採用するべきである。

6 まとめ

現在すでにコンシューマ向けの IPv6 アクセス網がサービスインしており、独自にインターネットを利用するソフトウェアや、組み込み機器の IPv6 対応も必要となりつつある。
これらのソフトウェア・ファームウェアにとって、IPv6 対応は従来なかった新しいプロトコルに対応することであり相応にコストがかかる。特に、IPv4 のみに対応していたシングルプロトコル構造に対して、IPv6 対応はマルチプロトコル構造に転換することを意味しており、ソフトウェア・ファームウェアの修正・追加量は増大することが予測される。
当該資料はそれらソフトウェア・ファームウェアを開発する際の、問題・課題・注意点をまとめたものである。
今後、新たな問題や課題・実装提案がされることもあるため、IPv6 ソケットプログラミングを行うソフトウェア・ファームウェア技術者・企画担当者は当該資料に加えて、インターネットの標準化動向を常にウォッチする必要があるだろう。
その助けになればと考え、またインターネットの健全な発展の助けになればと考え、関係諸氏のご協力を得て当該文書を作成した。

7 検討メンバー

下記に本書の検討メンバーを示す。会務担当者以外のメンバーは、氏名の 50 音順に従っている。

氏名	所属
荒野 高志（WG 主査）	IT ホールディングス株式会社
廣海 緑里（chair）	株式会社インテック
波田野 裕一（co-chair）	日本 UNIX ユーザ会(jus)
藤崎 智宏（co-chair）	NTT 情報流通プラットフォーム研究所
新 善文	アラクサラネットワークス株式会社
大平 浩貴	株式会社リコー
佐藤 良	株式会社コナミデジタルエンタテインメント
高田 美紀	株式会社 NTTPC コミュニケーションズ
渡辺 露文	富士ソフト株式会社
谷口 寛季	富士ソフト株式会社
花山 寛	ネットワンシステムズ株式会社