ソケットライブラリの改善
socket library improvement

田中 哲

産業技術総合研究所
2009-07-19

どんな改善?
What kind of improvement?

いろいろ
Variously

どんな改善?
What kind of improvement?

プロトコル非依存
Protocol Independent

どんな改善?
What kind of improvement?

IPv6

TCPServer はプロトコル依存
TCPServer is Protocol Dependent

- TCPServer.open(9999) {|serv|
  s = serv.accept
  ...
  }
- FreeBSD などでは IPv4 を扱えない
  doesn't work with IPv4 on FreeBSD, etc.
- IPv4 と IPv6 を両方とも扱うべき
  IPv4 and IPv6 should be handled simultaneously

ソケットライブラリの問題
socket library problems

- TCPServer, UDPSocket はプロトコル依存
  TCPServer and UDPSocket is protocol dependent
- 冗長な引数やわかりにくい値
  Redundant Arguments/Unclear Values
- 足りない機能: sendmsg/recvmsg など
  Lacked features: sendmsg/recvmsg, etc.

問題: プロトコル依存
Problem: Protocol Dependent

- IPv6 の推奨は、プロトコル非依存
  IPv6 recommends protocol independent
- TCPServer.open(9999) は IPv4 を受け付けないことがある
  TCPServer.open(9999) may refuse IPv4
- UDPSocket.new() では IPv4 しか動かない
  UDPSocket.new() works only IPv4
- UDPSocket.new(Socket::AF_INET6) は IPv6 しか動かない
  UDPSocket.new(Socket::AF_INET6) works only IPv6

問題: 冗長な引数・わかりにくい値
Problem: Redundant Args/Unclear Values

- UDPSocket.new(Socket::AF_INET6)
- Socket.new(Socket::AF_INET,
  Socket::SOCK_STREAM, 0)
- sock.getsockopt(Socket::SOL_SOCKET,
  Socket::SO_LINGER)
  #=> "¥x00¥x00¥x00¥x00¥x00¥x00¥x00¥x00"
- Socket.sockaddr_in(port, host)
  #=> "¥x02¥x00¥x00P¥xDD¥xBA¥xB8D¥x00¥x00..."
- Socket.getaddrinfo("www.ruby-lang.org", "http")
  #=> [["AF_INET", 80, "carbon.ruby-lang.org",
  "221.186.184.68", 2, 1, 6]]

問題: 足りない機能
Problem: Lacked Features

- ホストの IP アドレスを得る
  Obtain the IP addresses of the host
- sendmsg/recvmsg

どうやって解決するか?
How they are solved?

- プロトコル依存 → 新しいAPI
  Protocol dependent → New API
- 冗長な引数 → シンボルを受け付ける
  Redundant arguments → Accept symbols
- わかりにくい値 → inspect を持つ新しいクラス
  Unclear values → New class with inspect
- 足りない機能 → 新しい API
  Lacked Features → New API

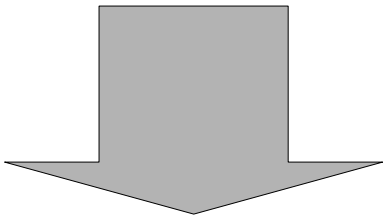
新しいAPI: Socket を使いやすく
New API: Make Socket Easy

- 高レベル API はすべてのソケットには対応できない
  High level API can't be generic socket
  - クラス名がソケットの種類を限定する
    The class names restricts a kind of socket
    TCPSocket, UDPSocket, etc.
- 低レベル API を使いやすくすることは可能
  It is possible to make low level API easier

冗長な引数・
わかりにくい値
Redundant Arguments/
Unclear Values

定数のプリフィクス
Prefix of Constants

- C:
  socket(AF_INET, SOCK_STREAM, 0)
- Ruby 1.9.1:
  Socket.new(Socket::AF_INET,
  Socket::SOCK_STREAM, 0)

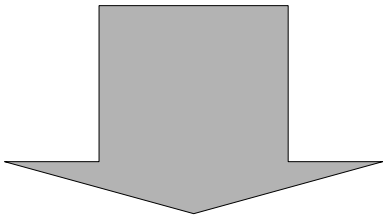


C より長いのはおかしい
It is wrong because longer than C

- Ruby 1.9.2:
  Socket.new(:INET, :STREAM)

Socket.getaddrinfo

- Ruby 1.9.1:
  Socket.getaddrinfo("www.ruby-lang.org", "http")
  #=>
  [["AF_INET", 80, "carbon.ruby-lang.org",
  "221.186.184.68", 2, 1, 6]]



Addrinfo クラスの導入
New Addrinfo class

- Ruby 1.9.2:
  Addrinfo.getaddrinfo("www.ruby-lang.org", "http")
  #=>
  [#<Addrinfo: 221.186.184.68:80 TCP (www.ruby-lang.org:http)>]

Addrinfo

- family + socktype + protocol + sockaddr
- プロトコル非依存なサービスのアドレス
  Protocol independent address for services
- C の getaddrinfo() が返す struct addrinfo に対応
  Similar to struct addrinfo returned by getaddrinfo() in C
- 中身を考慮したわかりやすい inspect
  Tailored inspect method
- connect, bind したソケットの生成メソッド
  connected/bound socket creation method

Addrinfo を使った接続
Connection using Addrinfo

```
a = Addrinfo.tcp("www.ruby-lang.org", "http")
a #=> #<Addrinfo: 221.186.184.68:80 TCP
              (www.ruby-lang.org:http)>
a.ip_address #=> "221.186.184.68"
a.ip_port #=> 80
a.to_s #=> "¥x02¥x00¥x00P¥xDD¥xBA¥xB8D..."
a.connect {|s|
  s.print "GET / HTTP/1.0¥r¥n"
  ...
}
```

バイナリな値
Binary Values

- struct sockaddr
  sock.getsockname
  #=> "¥x02¥x00¥xE2¥xB4¥x00¥x00¥x00¥x00..."

- socket option
  sock.getsockopt(Socket::SOL_SOCKET,
  Socket::SO_LINGER)
  #=> "¥x00¥x00¥x00¥x00¥x00¥x00¥x00¥x00"

sockaddr を Addrinfo でラップ
Wrap sockaddr as Addrinfo

struct sockaddr → Addrinfo

- sock.getsockname
  #=> "¥x02¥x00'¥x0F¥x7F¥x00¥x00¥x01¥x00..."

- sock.local_address
  #=> #<Addrinfo: 127.0.0.1:9999 TCP>

  family ← address family
  socktype ← getsockopt(SO_TYPE)
  protocol ← 0

socket option のラップ
Wrap socket option

socket option → Socket::Option

- 1.9.1

```
s = Socket.new(Socket::AF_INET,
                        Socket::SOCK_STREAM, 0)
s.getsockopt(Socket::SOL_SOCKET,
                      Socket::SO_TYPE)
#=> "¥x01¥x00¥x00¥x00"
```

- 1.9.2

```
s = Socket.new(:INET, :STREAM)
s.getsockopt(:SOCKET, :TYPE)
#=> #<Socket::Option: INET SOCKET TYPE SOCK_STREAM>
```

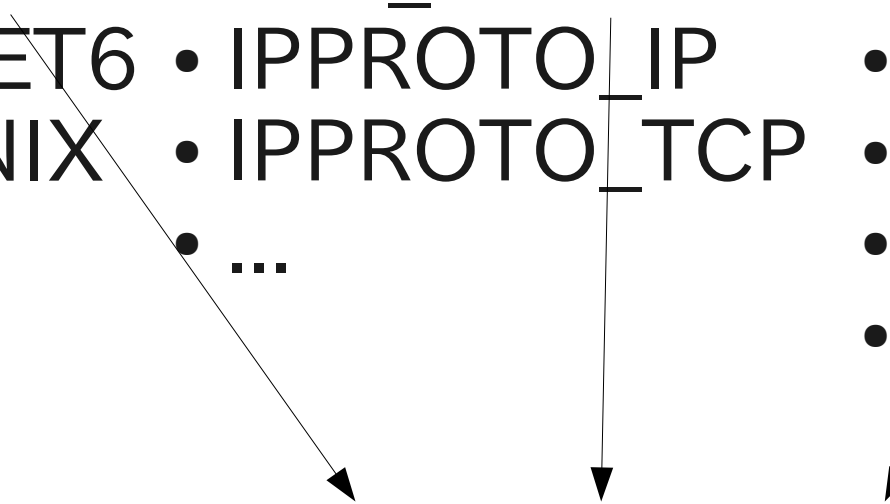
Socket::Option

Socket::Option = family + level + name + value

family
- AF_INET
- AF_INET6
- AF_UNIX
- ...

protocol level
- SOL_SOCKET
- IPPROTO_IP
- IPPROTO_TCP
- ...

option name
- SO_TYPE
- SO_REUSEADDR
- IP_OPTIONS
- TCP_NODELAY
- ...



#<Socket::Option: INET SOCKET TYPE SOCK_STREAM>

"¥x01¥x00¥x00¥x00" = SOCK_STREAM

If SOL_SOCKET, SO_TYPE, little endian, 32bit int

TCPServer, UDPSocket はプロトコル依存
TCPServer and UDPSocket is protocol dependent

プロトコル非依存
Protocol Independent

- IPv4 と IPv6 をいっしょに扱う
  Handle IPv4 and IPv6 together
- 概念的には、IPv4, IPv6 以外も扱う
  In general, handle protocols other than IPv4, IPv6 too
- クライアントは IPv4 と IPv6 の両方を試す
  Clients try both IPv4 and IPv6
- サーバは IPv4 と IPv6 の両方で待つ
  Servers waits connections from IPv4 and IPv6

プロトコル非依存 TCP サーバ
Protocol independent TCP server

- Socket.tcp_server_loop(9999) {|s|
  # s is an accepted socket
  }
- TCPServer.open(9999) はポータブルでない
  TCPServer.open(9999) is not portable
  - 4.4BSD では IPv4 を扱えない
    doesn't work with IPv4 on 4.4BSD
    - FreeBSD
    - NetBSD
    - OpenBSD
  - Windows XP でも
    On Windows XP too

IPv6 の TCP サーバ
TCP server with IPv6

- IPv6 のソケットは IPv4 を扱えるか?
  Can IPv6 socket handle IPv4?
  - Yes, by default
    - GNU/Linux
    - Mac OS X
    - Solaris
  - No, by default
    - NetBSD
    - FreeBSD
  - No, always
    - OpenBSD
    - Windows XP

セキュリティ上の理由
Because security reason

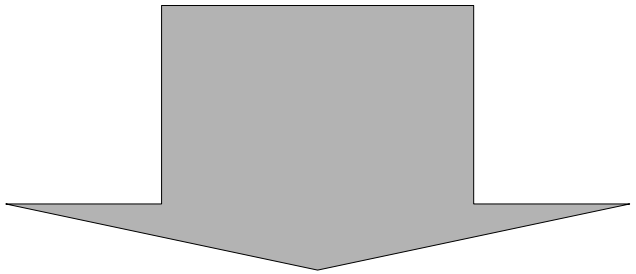
ポータブルな TCP サーバ
Portable TCP server

- IPv4 と IPv6 のふたつのサーバソケットを使う
  Use two server sockets for IPv4 and IPv6
- IPv6 のソケットは IPv4 を扱わないように設定
  Force the IPv6 socket don't handle IPv4 (IPV6_V6ONLY socket option)
- select() を使って両方で待つ
  Wait the both sockets using select()

TCPServer の問題と解決
TCPServer problem and solution

- TCPServer.open(9999) {|serv|
  loop {
    s = serv.accept
    ...
  }
}

ひとつのサーバソケット
Single server socket



- Socket.tcp_server_loop(9999) {|s|
  ...
}

サーバソケットは隠蔽
Server sockets hidden

UDP ソケットの問題
Problems of UDP socket

- プロトコル非依存な UDP は難しい
  Protocol independent UDP is difficult
- マルチホーム環境で、届いたアドレスから返したい
  Reply packets from received address on multihomed environment

プロトコル非依存な UDP
Protocol independent UDP

- プロトコル非依存な UDP は難しい
  Protocol independent UDP is difficult
  - サーバは IPv4, IPv6 両方で待てば可能
    Server is possible if wait IPv4 and IPv6
  - クライアントはうまくいかない
    Client doesn't work well

マルチホーム環境の UDP
UDP on Multihome Environment

- マルチホーム環境で、届いたアドレスから返したい
  Reply packets from the received address on multihomed environment
  - IPv6 では recvmsg/sendmsg で IPV6_PKTINFO
    recvmsg/sendmsg with IPV6_PKTINFO for IPv6
  - IPv4 では IP アドレス毎にソケットを作る
    socket for each IP address for IPv4

プロトコル非依存 UDP サーバ
Protocol independent UDP server

- Socket.udp_server_loop(port) {|msg, msg_src|
  msg_src.reply(msg)
  }
- msg_src は送信元アドレスと宛先アドレスを保持
  msg_src contains source and destination address
- reply メソッドは宛先アドレスから引数を送信する
  reply method sends the arguments from the destination address

プロトコル非依存UDPクライアント
Protocol Independent UDP client

- connect できても実際にサーバがあるか不明
  server availability is unknown even if connect succeed
- サーバが存在するかどうか判断するのもアプリケーション依存
  It depends on application to test server availability
- プロトコルを隠蔽できない
  Cannot hide protocol dependency

プロトコル非依存 TCP クライアント
Protocol Independent TCP client

- TCPSocket はほぼ問題ない
  Almost no problem with TCPSocket
- Socket でプロトコル非依存にやるのは面倒
  It is hard to use Socket for TCP with protocol independent style
- Socket.tcp を用意する
  New method Socket.tcp

  Socket.tcp(host, port) ≒
  TCPSocket.open(host, port)

足りない機能:
sendmsg/recvmsg など
Lacked features:
sendmsg/recvmsg, etc.

足りない機能
Lacked Features

- ホストの IP アドレスを得る
  Obtain the IP addresses of the host
- sendmsg/recvmsg

ホストの IP アドレス
IP Addresses of the Host

- Socket.ip_address_list
  #=> [#<Addrinfo: 127.0.0.1>,
  #<Addrinfo: 221.186.184.67>,
  #<Addrinfo: ::1>,
  #<Addrinfo: fe80::211:43ff:fefd:66%eth0>]
- IPv4 の UDP で宛先アドレスを得るのに必要
  Required for destination address of IPv4 UDP
- IPv6 の接続性の判断にも使える
  Usable to determine IPv6 connectivity

IPv4 の UDP で宛先アドレス
Destination Address of IPv4 UDP

マルチホーム環境の UDP サーバ

UDP server on multi homed environment

- IP アドレス毎にソケットを作って bind する
  create and bind for each IP address
- パケットの宛先は到着したソケットから判明する
  The destination of a packet can be determined by socket
- 返事は到着したソケットから行う
  Reply using the socket which message arrives
- 返事の送信元は元のメッセージの宛先になる
  The source of the reply will be the destination of original message

IP アドレスを得る方法
How to obtain IP addresses

- 残念ながら標準化されていない
  No standard, sigh
- 実装方法
  Implementation
  - getifaddrs: BSD/OS, FreeBSD, NetBSD, OpenBSD, DragonFly BSD, MirOS BSD, GNU/Linux, MacOS X, AIX
  - SIOCGLIFCONF: Solaris
  - SIOCGIFCONF: 4.3BSD (No IPv6)
  - GetAdaptersAddresses: Windows

sendmsg/recvmsg

- 補助的なデータを受け渡せる send, recv
  send and recv with ancillary data
- 補助的なデータ:
  ancillary data:
  - rights: file descriptor passing
  - time stamp: When UDP packet arrived?
  - credential: Who sends the packet?
  - destination address
  - IPv6 でいろいろつかわれる
    Extensively used with IPv6

Ruby で sendmsg/recvmsg
sendmsg/recvmsg on Ruby

- Socket::AncillaryData で補助データを表現
  New Socket::AncillayData class
- Addrinfo.udp("0.0.0.0", 9999).bind {|s|
  s.setsockopt(:SOCKET, :TIMESTAMP, 1)
  p s.recvmsg
  }

request packet arrival time

#=>
["a", #<Addrinfo: 127.0.0.1:50309 UDP>, 0,
 #<Socket::AncillaryData: INET SOCKET TIMESTAMP
  2009-07-15 00:50:11.793562>]

packet arrival time

IPv6 で到着したアドレスから返事
Reply from dest. address in IPv6

- IPV6_PKTINFO を使う
- Addrinfo.udp("::", 9999).bind {|s|

```
  s.setsockopt(:IPV6, :RECVPKTINFO, 1)
  msg, src, flags, *ctls = s.recvmsg
  p [msg, src, flags, *ctls]
  s.sendmsg(msg, 0, src, *ctls)
}
#=>
["a",
 #<Addrinfo: [::1]:55150 UDP>, 0,
 #<Socket::AncillaryData: INET6 IPV6 PKTINFO ::1 lo>]
```

request pktinfo

receive pktinfo

specify pktinfo

宛先が ::1 のパケットが lo インターフェースに到着
The packet which dest. address is ::1 is arrived to lo interface

プロトコル非依存 UDP サーバ
Protocol Independent UDP Server

- Socket.udp_server_loop の実装:
  The implementation of udp_server_loop
- IPv4 は IP アドレス毎にソケットを作る
  Create a socket for each IP address for IPv4
- IPv6 は IPV6_PKTINFO を使う
  Use IPV6_PKTINFO for IPv6
- 到着したアドレスから返事をする
  Reply from the destination address

まとめ
Summary

改善 (1)
improvement (1)

- 新しいメソッド:
  New methods:

Socket.tcp
Socket.tcp_server_loop
Socket.tcp_server_sockets
Socket.unix
Socket.unix_server_loop
Socket.unix_server_socket
Socket.accept_loop
Socket.ip_address_list

BasicSocket#local_address
BasicSocket#remote_address
BasicSocket#connect_address
BasicSocket#sendmsg
BasicSocket#sendmsg_nonblock
BasicSocket#recvmsg
BasicSocket#recvmsg_nonblock
BasicSocket#getpeereid
Socket#ipv6only!

- 新しいクラス:
  New classes:
  - Addrinfo
  - Socket::Option
  - Socket::AncillaryData

改善 (2)
improvement (2)

- 引数を簡潔に受けつけるようにした:
  More succinct arguments:
  Socket.new(Socket::AF_INET, Socket::SOCK_STREAM, 0)
  →Socket.new(:INET, :STREAM)
- バイナリをラップ (ちょっと非互換):
  Change binary to object (bit incompatible):
  udpsock.recvfrom(100)
  #=>
  ["a", "¥x02¥x00¥x98r¥x7F¥x00¥x00¥x01¥x00¥x00¥x00¥x00¥x00¥x00¥x00¥x00"]
  →
  ["a", #<Addrinfo: 127.0.0.1:39026 UDP>]

まとめ
Summary

- プロトコル非依存性が上がった
  More Protocol Independent
- Socket が使いやすくなった
  Easier Socket class
- OS のいろんな機能を使えるようになった
  Support various OS features
- Socket::Option などは IPv6 と関係なく便利
  Socket::Option, etc. are useful regardless of IPv6