IPv6 support for SML/NJ

Katsuhiro Ueno

katsuu@jaist.ac.jp

Japan Advanced Institute of Science and Technology
Backgrounds

- SML/NJ lacks APIs for IPv6 at all
- We can’t IPv6 network programming with SML/NJ

Need to add IPv6 support to SML/NJ
Contents

- What is SML/NJ?
- Overview of socket implementation in SML/NJ
- What I did
- Demonstration
- Appendix
What is SML/NJ?
What is SML/NJ?

- Stands for Standard ML of New Jersey
- An implementation of Standard ML
- ML is a strongly typed functional programming language
- The latest stable version was released on September 2000.

http://www.smlnj.org/
Structure of SML/NJ

- Human
  - Interactive Interface
  - Compiler
  - Backend
  - Standard ML Basis Library

- Written in ML

- Written in C
  - Runtime System

- Operating System

- SML/NJ

IPv6 support for SML/NJ – p.5/24
Socket implementation in SML/NJ
Points to be checked

- How to deal with socket APIs in C
- How to deal with socket addresses in runtime
- Structure of user-level APIs in Basis Library
- How to deal with sockets and socket addresses in ML
How to deal with socket APIs

The diagram illustrates the interaction between different components in dealing with socket APIs:

1. **Human** interacts with the **Interactive Interface**.
2. The **Interactive Interface** calls the **Compiler**.
3. The **Compiler** compiles the **structure Socket**.
4. The **Standard ML Basis Library** makes up a module - defines data types and adds utility functions.
5. The **Runtime System** wraps the **socket(2)** and **bind(2)** calls.
6. The **Backend** connects to the **Operating System**.

The diagram shows the process of integrating socket APIs into the Standard ML system, from human interaction to operating system calls. IPv6 support for SML/NJ is mentioned on page 8/24.
How to deal with socket addresses

- Runtime simply maps a `sockaddr` structure in C into an octet array object in ML.
- Protocol independent
User-level APIs for socket

Basis Library provides several structures for socket.

- Socket structure
- INetSock structure
- UnixSock structure
- GenericSock structure
- NetHostDB structure
- NetProtDB structure
- NetServDB structure
Types for socket

- Sockets are typed with respect to each combination of
  - address family (AF_INET, AF_UNIX, ...),
  - socket type (SOCK_DGRAM, SOCK_STREAM, ...),
  - status of socket (passive, active, ...)

- Socket addresses are typed WRT each address family.

\[\downarrow\]

Inhibits protocol-independent socket programming!
- let val is = INetSock.TCP.socket()
  = val us = UnixSock.DGrm.socket()
  = in [is, us] end;
Error: operator and operand don’t agree
operator domain:
  'Z INetSock.stream_sock
  * 'Z INetSock.stream_sock list
operand:
  'Z INetSock.stream_sock
  * UnixSock.dgram_sock list
What I did
What I did

- Design
- Implementation
- Release and contribution
- Making example
Summary of design

- INet6Sock structure
  - IPv6 specific functionalities
    - `inet_ntop(3), inet_pton(3), ...`

- NetAddrDB structure
  - protocol-independent name resolution
  - protocol-independent socket creation
    - `getaddrinfo(3), getnameinfo(3), ...`
Point of design

- Socket of `AF_INET6` is typed as 
  `'st INet6Sock.sock`.

- The result of `getAddrInfo` is typed in protocol-independent manner.
  - Defer concrete type generation as long as possible

- Never forget `get/setIPV6ONLY`
Implementation

- SML/NJ doesn’t use GNU autoconf
  - Compile with -DIPV6 flag to enable IPv6 support
- Absorb subtle difference between implementations of `getaddrinfo`
- `get/setIPV6ONLY` never fail even if OS doesn’t provide it
Difference of getaddrinfo(3)

If `AI_CANONNAME` is specified,
- set same string as given hostname to `ai_canonname` (RFC3493, KAME, Solaris)
- do reverse resolution (glibc, old KAME)
- set NULL to `ai_canonname` if resolution is failed
- fill all `ai_canonname` fields (KAME)
- not fill all `ai_canonname` fields (RFC3493, Darwin)

↓

Make sure all `ai_canonname` are filled
Release and contribution

You can obtain all materials from:

http://www.jaist.ac.jp/~katsuu/smlnj-ipv6/
Example

Simple HTTP server written in Standard ML
- create IPv4 and IPv6 socket separately
- use `select(2)` to deal with two passive sockets

Be careful for types of sockets!
Demonstration
Appendix
Bugs of Mac OS X 10.3

- `getnameinfo(3)` fails when translating specific port into service name
- `inet_ntop(3)` may cause buffer overflow

I sent reports with bugfix patches to Apple (Problem ID #3973768, #3972456)
Thank you for listening.