Status of this Memo

This Internet-Draft is submitted to IETF in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt.

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

This Internet-Draft will expire on February, 2010.

Copyright Notice

Copyright (c) 2009 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents in effect on the date of publication of this document (http://trustee.ietf.org/license-info). Please review these documents carefully, as they describe your rights and restrictions with respect to this document.
Abstract

The Socket API is the de facto standard of network API for many UNIX-like and non-UNIX OSes. Many APIs that are exported to the user space is either direct use or encapsulates the Socket API. The Socket API has been extended to support IPv6. Recently, multiple interfaces or multiple connection capable devices become more and more common. But due to the limitation of the default route of the host and other issues such as DNS selection etc prevent the utilization of multiple interfaces/connections benefit. Moreover, there is no API level support for the application developer to utilize the benefit of the host’s multiple interfaces/connections. Starting with the requirement of MIF API extension, this document introduces a new set of APIs to support multiple interfaces/connections.

Table of Contents

1. Introduction................................................3
2. Conventions used in this document..........................3
3. API Extension Requirement..................................3
   3.1. R1: Compatible with existing Socket APIs.................3
   3.2. R2: Support for Multiple Connections....................3
   3.3. R3: No Changes to Existing Socket APIs..................3
   3.4. R4: Easy for Implementation...........................3
4. Data structure need to be extended.........................3
5. Definition of New Socket APIs................................3
   5.1. Physical interface/connection status acquiring API......3
   5.2. Connection mapping API..................................3
   5.3. Connection status query API..............................3
   5.4. Socket and GenericConnection bind API...................3
6. Using the extended APIs....................................3
   6.1. An Example.............................................3
   6.2. Implementation Details................................3
7. Security Considerations......................................3
8. IANA Considerations........................................3
9. Acknowledgments............................................3
10. References................................................3
    10.1. Normative References................................3
    10.2. Informative References...............................3
Author’s Addresses...........................................3
1. Introduction

Socket API was first used in UNIX operating system as network API for application developers to develop network applications. Then it is introduced to many other operating systems and become a de facto standard of network APIs.

The Socket API needs to be extended according to the evolution of TCP/IP protocol stack. For example, the Socket API was extended to make it support IPv6 in RFC 3493 [RFC3493]. Nowadays, there are more and more devices that have multiple interfaces/connections, but due to the limitation of Socket API and other issues such as default route and DNS selection, the application developer can not direct use the multiple interfaces/connection capability of the devices. This document aims to extend the Socket API to make it support multiple interfaces/connection while remain compatibility with current Socket.

2. Conventions used in this document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC-2119.

3. API Extension Requirement

This section starts with several requirements for MIF API extension.

3.1. R1: Compatible with existing Socket APIs

The extension of Socket APIs should remain compatible with current existing Socket APIs which means this extension will not affect the existing application and the application developer can chose whether to use this extension or not.

3.2. R2: Support for Multiple Connections

The newly introduced APIs can support the host applications to utilize multiple connections. If some connections fail to work, applications can failover to other available interfaces.

3.3. R3: No Changes to Existing Socket APIs

The newly introduced APIs cannot change the behavior of existing network APIs available on the host operating system.
3.4. R4: Easy for Implementation

The implementation details for the newly introduced APIs should be as neat as possible. It keeps the underlying implementation details simple, and makes the application developers easy to use these APIs.

4. Data structure need to be extended

This document introduces a new data structure named Generic_connection in addition to the current Socket API. This is an abstraction of the physical connections of the host. The definition of this data structure is as follows:

Struct GenericConnection {
    struct GenericConnection *next;
    int index;
    struct Netdevice *dev;
    int priority;
    char status;
    struct socket *sock;
}

This structure has many data items defined as follows:

- **next**: this is an pointer that points to another GenericConnection structure. This means that many GenericConnection structure can form a link list.

- **index**: the index indicates the position of a particular GenericConnection item of the GenericConnection structure link list.

- **dev**: this is a pointer that points to a particular physical interface structure.

- **Priority**: priority indicates the priority of the connection.

- **status**: status indicates the status of current connection.

- **Sock**: sock is a pointer that points to a socket structure.
5. Definition of New Socket APIs

To make it compatible with existing Socket API, this document will not define new parameters to existing Socket API. Instead, this document defines a set of new socket APIs that used to support multiple connections feature.

5.1. Physical interface/connection status acquiring API

For the host that has multiple interfaces/connections, by invoking this API, it can return the current interfaces/connections that the host has. The definition of this API is:

```
int ListPhysicalConnection(struct Netdevice *dev, int connectionNumber),
```

the parameter’s definition is:

- dev: this is a pointer that points to a link list of Netdevice structure.
- connectionNumber: this is the number of the physical connections.

5.2. Connection mapping API

To map a physical API to the GenericConnection structure, this subsection introduces a new API named ConnectionMap() to the current Socket API. The definition of this API is as follows:

```
int ConnectionMap(struct GenericConnection *G_connection, struct Netdevice *dev, int priority, struct socket *sock )
```

The definition of the parameters are:

- G_connection: this is a pointer that points to a GenericConnection link list;
- dev: this is a pointer that points to a physical connection.
- priority: this parameter defines the priority of this connection.
- sock: this is a pointer that points to a socket structure.

This API makes the mapping between the GenericConnection and the physical connections. Many physical connections can be mapped to one GenericConnection. For example, a host may has a WiFi interface and a Ethernet interface, the application developer can first invoke ListPhysicalConnection() to acquire the list of physical interfaces,
then the application can map both WiFi and Ethernet interfaces to the same GenericConnection structure.

Further more, by using this mechanism the OS vender can implement their own APIs that supports connection polices. For example, the OS vender can provide connection policy API to the application developer to enforce connection polices.

The application developer can also use these functions to implement their own connection polices directly. This design gives the flexibility of the implementation of connection policies.

5.3. Connection status query API

This subsection introduces a new API name ConnectionStatusQuery() used to query the status of the connections, the definition of this API is as follows:

ConnectionStatusQuery() function returns the current status of the GenericConnection, the definition of this API is as follows:

\[
\text{int ConnectionStatusQuery}(\text{struct GenericConnection *} \text{G\_connection}, \text{int index}, \text{int status}),
\]

the definition of the parameters are:

- \text{G\_connection} a pointer that points to a GenericConnection link list.
- \text{index} the index of the connection that the caller what to query
- \text{status} the status of the connection that the caller queries

When the physical connection’s status changes, the change will reflects by the parameter "status". By invoking this API, the application can monitor the status of the physical connections or when the physical connection status changes will notify the application by setting the "status" parameter to the according value.

5.4. Socket and GenericConnection bind API

This subsection introduces an API to bind the Socket structure to the GenericConnection structure. The definition of this API is as follows:

\[
\text{BindToConnection}(\text{Struct socket *} \text{sock}, \text{Struct GenericConnection *} \text{G\_Connection} ),
\]

the definition of the parameters are:

- \text{sock} a pointer that points to a socket structure.
6. Using the extended APIs

6.1. An Example

To enjoy the benefit of multiple interfaces, the developers can write programs according to the APIs introduced in this document.

For example, if the application wants to bind to both the Ethernet and WiFi interfaces, it can follow the following steps:

1. create a GenericConnection variable, say gconn;

2. create two sockets binding to these two interfaces separately, sock_wired and sock_wifi;

3. use the connection mapping API introduced in Sec.5.2 to map the sock_wired and sock_wifi to the generic connection struct *gconn;

4. now that gconn consists of a list of sockets that maps to the wired socket and wireless socket separately, the application can start data connection using the generic connection class. And the implementation details of how the generic connection class utilizes those bound interfaces are left to the next section.

6.2. Implementation Details

The implementation details of these newly introduced APIs are to be introduced later.

7. Security Considerations

TBD

8. IANA Considerations

TBD

9. Acknowledgments

The authors want to thank Teemu Savolainen from Nokia, Dayi Zhao from Bitway and others for their useful suggestions and discussions.
10. References

10.1. Normative References


10.2. Informative References

Author’s Addresses

Dapeng Liu
China Mobile research institute
Unit2, 28 Xuanwumenxi Ave,Xuanwu District,
Beijing 100053, China

Email: liudapeng@chinamobile.com

Zhen Cao
China Mobile research institute
Unit2, 28 Xuanwumenxi Ave,Xuanwu District,
Beijing 100053, China

Email: caozhen@chinamobile.com