NETLINK SOCKET PARSER

==========

1. About Me

==========

1.1 Personal Details

==========

- **Name:** Abhishek Tiwari
- **Email:** erabhishektiwarics@gmail.com
- **IRC nick on freenode:** abhishekcs1**0**
- **Phone:** (+91) 7479024291
- **Country of Residence:** India
- **Timezone:** IST (GMT + 0530)
- **Primary language:** English
- **Github Profile:** https://www.github.com/abhishekcs10

1.2 Background and Programming experience

==========

I am first year student of IIT Kharagpur, West Bengal, India, pursuing M.Tech, major in Computer Science and Engineering. I have been taking part in various online competitive programming challenges Hackerrank (handle-abhishekcs10), Codechef (handle-abhishekssj5), and am looking forward to make contribution to open source community with my abilities. I have been making college projects using Version Control System (git) and thus have basic knowledge about using it. For various programming projects I have done, that can be viewed on my Github Profile.

1.3 Academic Details, Previous Work, Internships

==========

I am studying following subjects in my Masters-

- Internet Architecture and Protocols
- Information Retrieval
- Natural Language Processing
- High Performance Computer Architecture
- High Performance Parallel Programming

I am working as a Teaching Assistant in IIT Kharagpur, Programming and Data Structures Lab.

During M.Tech I have worked on various system related assignments like coding a mini C-shell (the_c_shell), socket programming (Networking_in_c) in order to handle multiple connections, Recording swipe features from soft keyboard in Android (AffectSense), etc.

I have done internship in Cyberzone Technologies and worked on Linux Administration and Security which covers system administration. We worked on Red Hat Enterprise version 6 OS.

1.4 Contribution to Strace
==========

* In Progress: Implement -e trace=%statfs option for tracing statfs like syscalls.
* Accepted and Merged: alpha, mips: fix missing flags in stat related compatibility syscalls.
* Accepted and Merged: Update information on how to build strace from git repository.

Q. Any previous open-source projects (or even previous GSoC) you have contributed to?

A. Babel-python (Worked on documentation part, resolved Issue 463.)

==========

2. Project Proposal

==========

Netlink Socket Parser
Suggested by: Gabriel Laskar, Dmitry V. Levin

2.1 Project Summary

==========

Netlink is a network protocol that is used to communicate between the kernel and the userspace. For example, iproute use netlink in order to configure the network stack (interfaces, addresses, routes, etc). It is also used by the kernel to report hotplug events to the userland.
The goal here is to add support in strace to decode netlink packet structures in order to be able to debug and discover netlink messages.
In 2016, the base ground work was done, now it can be extended to support more netlink family protocols.

2.2 Work done

==========

There has already been work done to decode headers of netlink packet and SOCK_DIAG has been traced and decoded. A summary for work done is explained in subsequent headings.

2.3 Understanding API

==========

Following is the code for basic SOCK_DIAG request

```
1.  int fd = socket(AF_NETLINK, SOCK_RAW, NETLINK_SOCK_DIAG);
2.  struct {
3.      struct nlmsghdr nlh;
4.      struct netlink_diag_req ndr;
5.  } req = {
6.      .nlh = {
7.          .nlmsg_len = sizeof(req),
8.          .nlmsg_type = SOCK_DIAG_BY_FAMILY,
9.          .nlmsg_flags = NLM_F_DUMP | NLM_F_REQUEST
10.     },
11.     .ndr = {
12.         .sdiag_family = AF_NETLINK,
13.         .sdiag_protocol = NDIAG_PROTO_ALL,
```

```
14.         .ndiag_show = NDIAG_SHOW_MEMINFO
15.     }
16. };
17. sendto(fd, &req, sizeof(req), MSG_DONTWAIT, NULL, 0);
```

A netlink socket is created same way as other network socket defined in Line 1. Here the first parameter (address family) is always given as AF_NETLINK, and second parameter (type) is always given as SOCK_RAW. The third parameter i.e. protocol changes.

2.3.1 Netlink Protocols
==========

The main protocols that have to be worked upon are:

- NETLINK_ROUTE: for manipulating the route subsystem
- NETLINK_SOCK_DIAG: for debugging sockets
- NETLINK_NETFILTER: for netfilter (current Linux firewall)
- NETLINK_GENERIC: a generic protocol API

2.3.2 Sending and Receiving Message
==========

Netlink sockets are connectionless, and operate in much the same way UDP sockets do. Messages are sent to recipients on an open netlink socket via the sendto and sendmsg library calls. Messages are received by the recvfrom and recvmsg library calls.

2.3.3 The Netlink Socket Address Structure
==========

The netlink socket address address structure, named struct sockaddr nl is passed to all calls which send or receive netlink sockets. This structure both informs the kernel networking stack of a datagrams destination, and informs a user-space program of a received frames source.

The structure is defined as follows:

struct sockaddr nl
{
sa family t nl family;
unsigned short nl pad;
u32 nl pid;
u32 nl groups;
}

• nl family - This field defines the address family of the message being sent over the socket. This should always be set to AF NETLINK.
• nl pad - This field is unused and should always be set to zero.
• nl pid - This field is PID 3 of the process that should receive the frame being sent, or the process which sent the frame being received. Set this value to the PID of the process you wish to receive the frame, or to zero for a multicast message or to have the kernel process the message.
• nl groups - This field is used for sending multicast messages over netlink. Each netlink protocol family has 32 multicast groups which processes can send and receive messages on. To subscribe to a particular group, the bind library call is used, and the nl groups field is set to the appropriate bitmask. Sending multicast frames works in a similar fashion, by setting the nl groups field to an appropriate set of values when calling sendto or sendmsg. Each protocol uses the multicast groups differently, if at all, and their use is defined by convention.

The **sockaddr nl** structure is cast to a standard sockaddr structure and passed in as the appropriate parameter to the send and recv families of library calls.

```
/* from linux/netlink.h */
1. struct sockaddr_nl {
2.    __kernel_sa_family_t nl_family;
3.     [...]
4. }
```

In order to get information about netlink protocols, a SOCK_DIAG request is made and after the system call, information is set in **netlink_diag_msg** from where the protocol information is retrieved.
With help of above SOCK_DIAG parser has been created.

==========

2.4 Goals

==========

Upto May 30: Understanding the codebase to get started with netlink protocols as currently I have no experience with them. Get to know how the existing code works and what are the various various functionalities provided till now.

May 31- June 15: Get to know what additionals protocols have to be worked upon and adding decoders to the remaining netlink protocols. SOCK_DIAG output has to be updated in order to get more information about protocol. Handling of socket that are not binded is to be done.

June 16- 30 June: Adding decoder for first protocol would take some time however once implemented adding decoders for others are expected to be done in a flow. I will work on implementing NETLINK_GENERIC protocol API in order to decode the protocol information.

July 1-July 15: Adding test cases for the implemented netlink protocol. Improving over the implementation part as suggested by mentor and decoding next protocol i.e. NETLINK_NETFILTER. Will try to cover following:

- Try to handle a maximum of netlink attributes.
- Handle nested attributes.

July 16-July 24: The netlink route protocol will be covered as much as possible (There are 400 - 500 for route, will try to cover important ones first).

- Handle socket that has not been binded.
- Handle ancillary data at msg level (cmsg).

July 25-August 29: Handling netlink route protocol will take much of the time and most probably will require extra time for handling most of the route functionalities.First priority will be to handle and finalize the changes done and doing documentation of the implemented code. The netlink route protocol will be then covered as much as possible (There are 400 - 500 for route, will try to cover as much as possible).

Q. Do you plan to have any other commitments during SoC that may affect you work? Any vacations/holidays planned? Will you be available full time to work on your project? (Hint: do not bother applying if this is not a serious full time commitment)

Ans. I would be having a regular summer internship, however I assure to give atleast 40 hours a week.

==========

3. References

==========

1. Strace implementation for netlink http://blog.saruta.eu/netlink_strace.html.
2. Redhat documentation for NETLINK NETLINK.pdf.