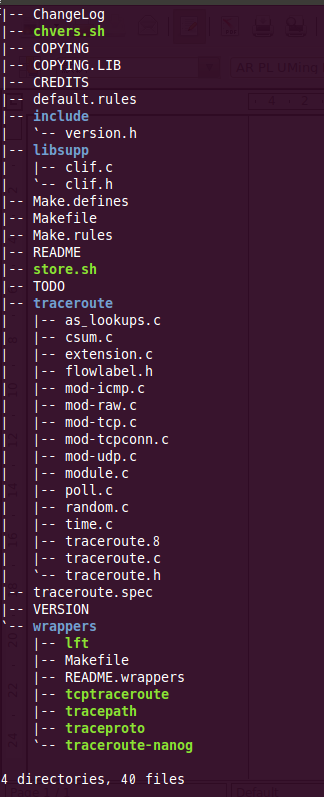
**增强型traceroute 实现**

GNU traceroute 的源码分析以及功能添加

汪宇 2706005012

准备工作

在GNU的官网上www.gnu.org 下载开源的GNU traceroute, 我使用的是 traceroute-2.0.16 , 压缩包的大小为63KB ，解压之后的大小为216.1KB ，以下是解压之后的目录文件



外围的文件中有两个 shell script文

件， chvers.sh , store.sh , 这两件文

会被Makefile所调用 ,另外Make.defines

,Make.rules , default.rules , traceroute.spec?都是和Makefile有关的 ， COPYING ， COPYING.LIB 介绍了GPL ，GUN General Public License (GNU 通用公共许可证）

CREDITS中作者对相关的人表示了感谢，

README中简单介绍了此traceroute的情况，

提到了这是为linux实现的最新的traceroute,以及其实一些主要特性，当然traceroute完整的功能描述可以在其man page 中看到

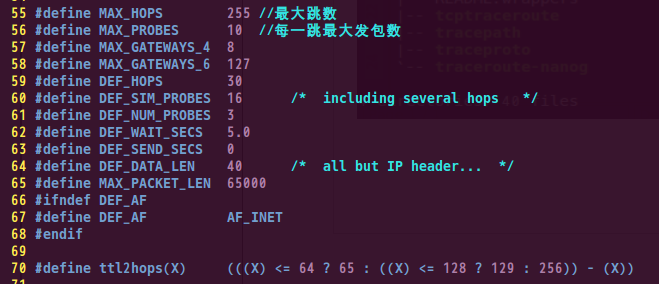
特别要提到TODO文件，软件的作者在里面提到了他对此程序的一些改进完善的想法，因为这是开源的GNU traceroute, 我们都可以自己动手去实现这些想法和我们自己的想法

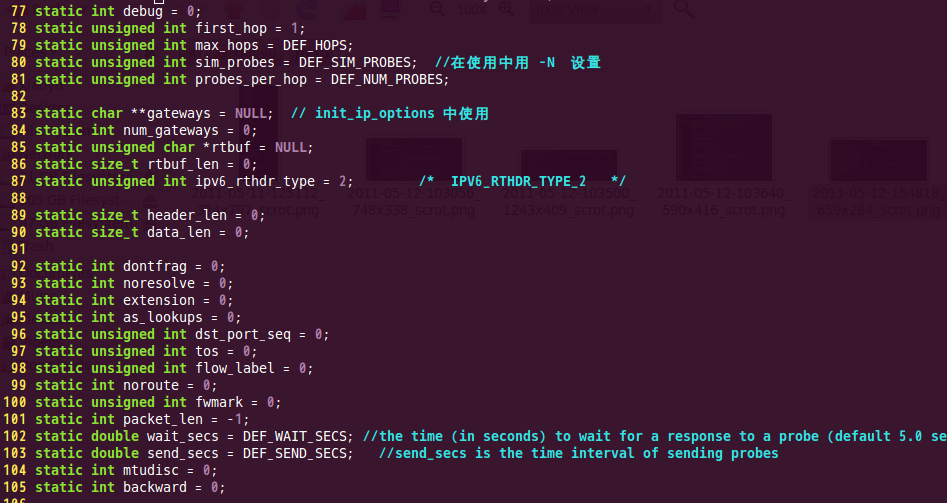
（这里还有 wrappers, libsupp..没有说）

我们可以看到，最主要的代码都在traceroute这个目录下，我们要深入剖析的也是这个目录下的代码

main函数定义在traceroute.c中，traceroute.c同时也定义了所有模块都要使用的公共函数，和相关的变量

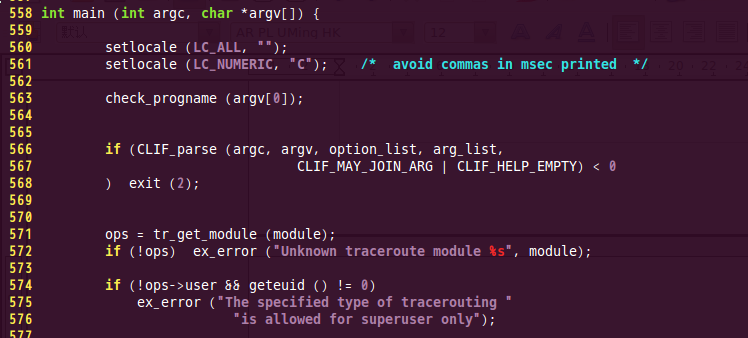
首先我们列出这个文件定义的宏

全局变量



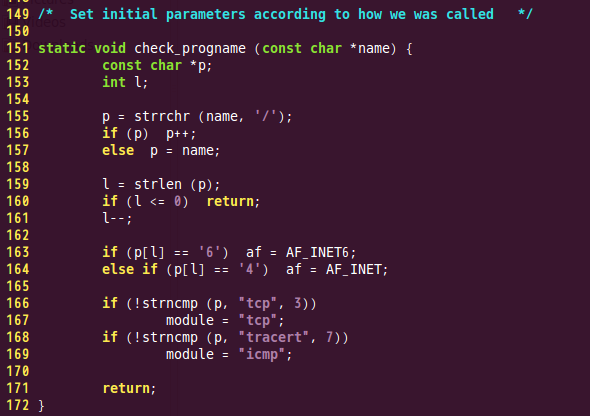
这些全局变量很重要，描述了程序和包的各个方面的性质，现在不必弄清楚这些变量是描述什么用的，在后面的函数分析过程中我们会反复提到这些变量，值得注意的是这些全局变量都定义为static,只在这一个文件中使用，non-static全局变量应该在头文件中定义

我们从main函数开始，首先要看的是命令行的分析



main函数开始几行定义如上， 第一二行是设置相关语言的

check\_progname 是这个在这个文件中自定义的一个函数



check\_progname(argv[0])对程序的调用名进行了检查，里面涉及到两个变量的设置 ，

af, module, 根据程序调用名 argv[0]对这两个变量进行了设置

从 ..图我们可以看到module定义 static const char \* module = “default”

module是一个字符串，用它来代表发包所用的模块

例如若名字为traceroute4则用ipv4发包，若名字为tcptraceroute则用tcp 模块发包，这时 module =“tcp”, 在我自己的电脑上，ubuntu系统上，不同的名字像tcptraceroute都是一个link file,指向唯一的 traceroute,然后再在程序里面来判断怎么发包

下面我们看566-568行，

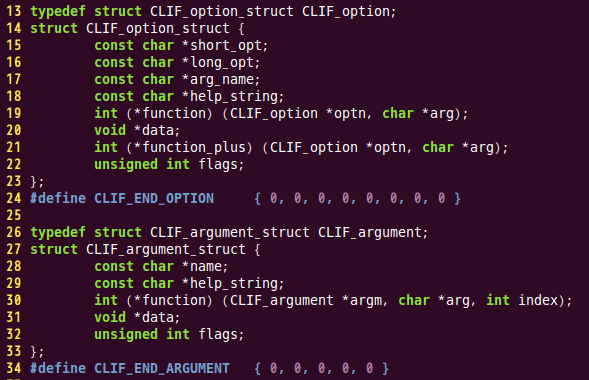
566 if (CLIF\_parse (argc, argv, option\_list, arg\_list,

567 CLIF\_MAY\_JOIN\_ARG | CLIF\_HELP\_EMPTY) < 0

568 ) exit (2);

这个函数是在/usr/include/clif.h 头文件中声明的 ，（应该并不是属于某个标准的接口函数，能找到的资料很少） ，这个文件并不是GNU C library的一部份

在 clif.h中定义了两个struct



在traceroute.c中定义了一个 struct CLIF\_option的数组，对每个命令行选项可能出现的参数都有了说明，及相关的函数调用和变量设置

举一个例子 在shell中敲下如下命令调用时

traceroute -f 3 [www.renren.com](http://www.renren.com/)

-f表示 first\_ttl 设为3,即从路径上的第三个路由开始记录，程序中相关的代码为

450 { "f", "first", "first\_ttl", "Start from the %s hop (instead from 1)",

451 CLIF\_set\_uint, &first\_hop, 0, 0 },

可以看到，若用了-f ，则会调用CLIF\_set\_uint 把 first\_hop的值设为我们-f后指定的值

当然调用的函数也可以是我们自己定义的函数

比如

452 { "g", "gateway", "gate", "Route packets through the specified gateway "

453 "(maximum " \_TEXT(MAX\_GATEWAYS\_4) " for IPv4 and "

454 \_TEXT(MAX\_GATEWAYS\_6) " for IPv6)",

455 add\_gateway, 0, 0, CLIF\_SEVERAL },

-g 设用来设置我们的包所必须经过的路由， add\_gateway 就是我们自己定义的函数， 这个在后面我们会说到

（我打算用getopt 从新改写 跟interface 有关的代码）

下面我们看traceroute.c中定义的两个出错处理函数，程序中到处都在用到

131 static void ex\_error (const char \*format, ...) {

132 va\_list ap;

133

134 va\_start (ap, format);

135 vfprintf (stderr, format, ap);

136 va\_end (ap);

137

138 fprintf (stderr, "\n");

139

140 exit (2);

141 }

142

143 void error (const char \*str) {

144

145 fprintf (stderr, "\n");

146

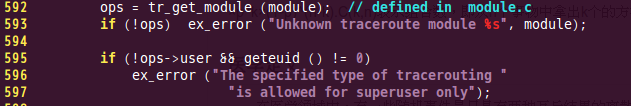
147 perror (str);

148

149 exit (1);

150 }

接下来的几行代码

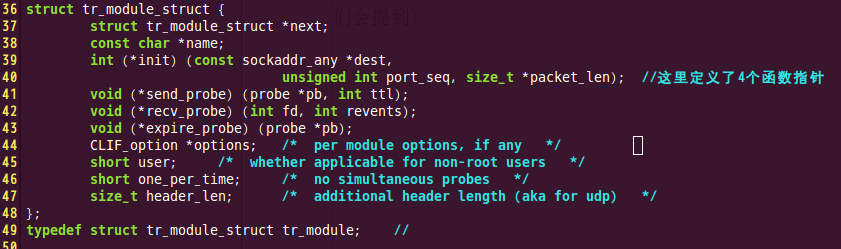


ops = tr\_get\_module (module) , 是非常关键的 ， 用 全局变量的定义中 我们可以看到114 static const tr\_module \*ops = NULL;

ops 是一个 struct tr\_module 型 的变量 ， tr\_module是在 traceroute.h中定义的一个struct，是整个程序中最重要的两个struct之一 （另一个是描述 发包探针的，后面我们会提到）

下面介绍一下这个 struct，tr\_module是用来描述每个发包模块的，每次我们调用traceroute，都会决定用哪一个发包模块来发包，是发UDP包（默认），TCP包（-T），还是ICMP包（-I）

tr\_module第一个成员变量是指向另一个tr\_module的pointer,这形成了链表，每一个我们程序所实现的module在这个链表中都会有一个struct与之对应，而这个链表在 main函数调用之前就已经形成了，下面我们会说这是怎么做到的

tr\_module别外的成员有 const char \* name , 这是module的名字，也就是traceroute.c中全局变量 module所对应的值，在我们的程序中，程序定义了6个module struct ,name 分别为 default , udp , udplite, tcp ,tcpconn ,icmp ,raw

分别在 mod-udp.c , mod-tcp.c , mod-tcpconn,c mod-icmp.c 中

45行有一个成员 short user; 表示是否需要root权限来运行这个 module ,

上面我们提到的6个module中，前三个的user 为1, 不需要root权限，后三个user为0,需要root权限

tr\_module还定义的4个函数指针，init, send\_probe, recv\_probe, expire\_probe分别指完成相应该功能的函数，这些函数在定义相应tr\_module的文件中定义，这些函数完成初始化，发包，收包，超时处理，非常重要，后面我们会看到

short one\_per\_time 的值都为1,表示每个probe所发的包个数

header\_len …??

CLIF\_option 是每个module的option,（后面对tcp举例）

其实在main函数调用之前，module.c 中的，tr\_register\_module就已经被调用过好几次了，这个函数是把每个mod-\*.c中定义的各自的tr\_module放到一个链表中，这个链表的表末是 NULL （module.c)中定义， 下面是摘自 module.c中的代码

16 static tr\_module \*base = NULL;

17

18 void tr\_register\_module (tr\_module \*ops) {

19

20 ops->next = base;

21 base = ops;

22 }

而这个函数是怎么定义的呢，下面我们以 default 模块举例，default module是我们traceroute 默认使用的 module,

定义如下， 代码摘自 mod-udp.c

203 static tr\_module default\_ops = {

204 .name = "default",

205 .init = udp\_default\_init,

206 .send\_probe = udp\_send\_probe,

207 .recv\_probe = udp\_recv\_probe,

208 .expire\_probe = udp\_expire\_probe,

209 .user = 1,

210 .header\_len = sizeof (struct udphdr),

211 };

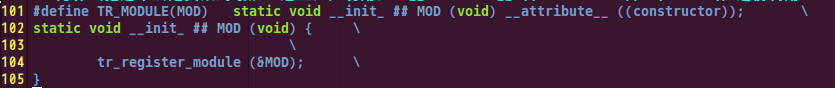
212

213 TR\_MODULE (default\_ops);

我们可以看到，default\_ops是一个static 全局变量，且其大部分成员已经初使化（struct 以如此方式初使化部分成员时名字前面加一个点.）

关健在于 宏 TR\_MODULE (default\_ops)

TR\_MODULE 这个宏是在traceroute.h的最后定义的(由于直接复制过来的代码不好看，这里用截图）



我们可以看到，宏TR\_MODULE展开后是一个函数声明和一个函数定义，以TR\_MODULE（default\_ops)举例

我们用命令 gcc -E -o mod-udp.i mod-udp.c

为mod-udp.c生成预处理后的文件，来查看宏展开后的样子：

A description...

我们可以清晰的看到生成了一个函数声明和一个函数定义，这个函数声明后后面加了\_\_attribute\_\_ ((constructor)) ， 表示这个函数在main函数调用之前就要调用

而在每个 tr\_module定义后面都有这样一个相应该的宏调用，用tr\_register\_module把所有定义的tr\_module组成一个链表，这一切都在main 函数调用之前发生

ops = tr\_get\_module(module)就是通过下面在module.c中定义的函数来得到我们此次调用程序将要使用的模块

24 const tr\_module \*tr\_get\_module (const char \*name) {

25 const tr\_module \*ops;

27 if (!name) return 0;

28

29 for (ops = base; ops; ops = ops->next) {

30 if (!strcasecmp (name, ops->name))// compare ingoring letter case

31 return ops;

32 }

33

34 return NULL;

35 }

（这里试着通过mod-udp.o来显示在main调用之前调用以上函数）