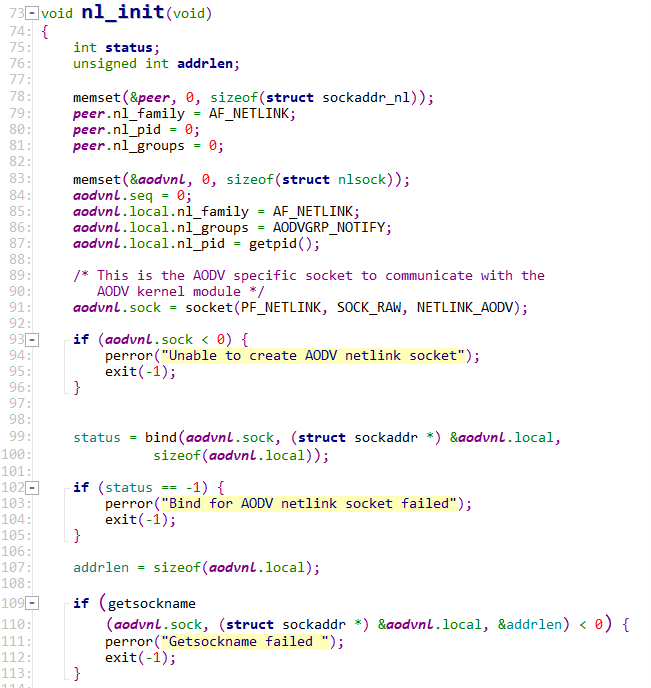
# 总览

|  |  |
| --- | --- |
| [void nl\_init(void)](#_void_nl_init(void)) | 建立与内核模块进行通信以及传递路由消息的套接字 |
| void nl\_cleanup(void) | 关闭套接字 |
| [static void nl\_kaodv\_callback(int sock)](#_static_void_nl_kaodv_callback(int) | 根据从与内核模块进行通信的套接字接收到的消息类型选用合适的回调函数 |
| [static void nl\_rt\_callback(int sock)](#_static_void_nl_rt_callback(int) | 根据从与内核模块传递路由的套接字接收到的消息类型选用合适的回调函数 |
| [int prefix\_length(int family, void \*nm)](#_int_prefix_length(int_family,) | 根据参数指定的地址族和IP地址计算出子网掩码长度即前缀长度 |
| [int addattr(struct nlmsghdr \*n, int type, void \*data, int alen)](#_int_addattr(struct_nlmsghdr) | 用于更新消息，增加一个属性 |
| [int nl\_send(struct nlsock \*nl, struct nlmsghdr \*n)](#_int_nl_send(struct_nlsock) | 把 netlink 消息嵌入到一个 iovec 结构体中然后用这个结构体构建msghdr消息，并将该消息发送给内核。 |
| [int nl\_kern\_route(int action, int flags, int family,int index, struct in\_addr \*dst, struct in\_addr \*gw,struct in\_addr \*nm, int metric)](#_int_nl_kern_route(int_action,) | 把 netlink 消息嵌入到一个 iovec 结构体中然后用这个结构体构建msghdr消息，并将该消息发送给内核。 |
| [int nl\_send\_add\_route\_msg(struct in\_addr dest, struct in\_addr next\_hop,int metric, u\_int32\_t lifetime, int rt\_flags,int ifindex)](#_int_nl_send_add_route_msg(struct_in) | 用来添加、移除、更新内核路由表中表项的函数 |
| [int nl\_send\_no\_route\_found\_msg(struct in\_addr dest)](#_int_nl_send_no_route_found_msg(stru) | 向内核发送一条netlink消息，将到达dest的路由信息设置为没有该条路由。 |
| [int nl\_send\_del\_route\_msg(struct in\_addr dest, struct in\_addr next\_h](#_int_nl_send_del_route_msg(struct_in) | 向内核发送消息将areq结构体中的路由m按照头部n的指示进行删除，随后调用nl\_kern\_route函数对路由表项进行删除 |
| [int nl\_send\_conf\_msg(void)](#_int_nl_send_conf_msg(void)) | 通过套接字向内核发送类型为配置消息的areq消息。 |

# 分析

## void nl\_init(void)



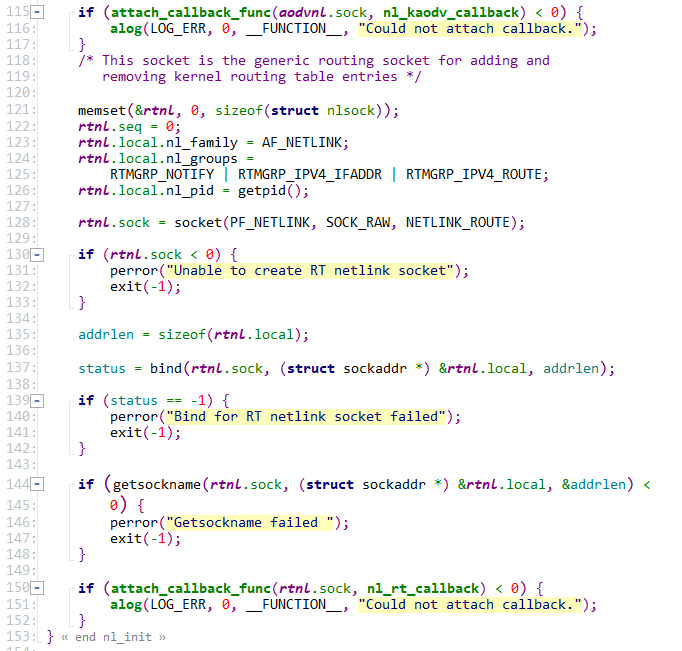
函数的前半部分

78~81 peer是一个sockaddr\_nl结构的实例，进行初始化为0操作后，对其进行各项参数填充，nl\_pid为进程id，nl\_groups为多播组掩码。

83~87 nlsock为自定义的结构体，aodvnl为一个nlsock的实例，同样初始化为0后进行其参数填充。

91 创建AODV专用套接字，指定协议族、套接字类型、使用协议，该套接字用来与内核模块进行通信

99~100 进行套接字的地址绑定

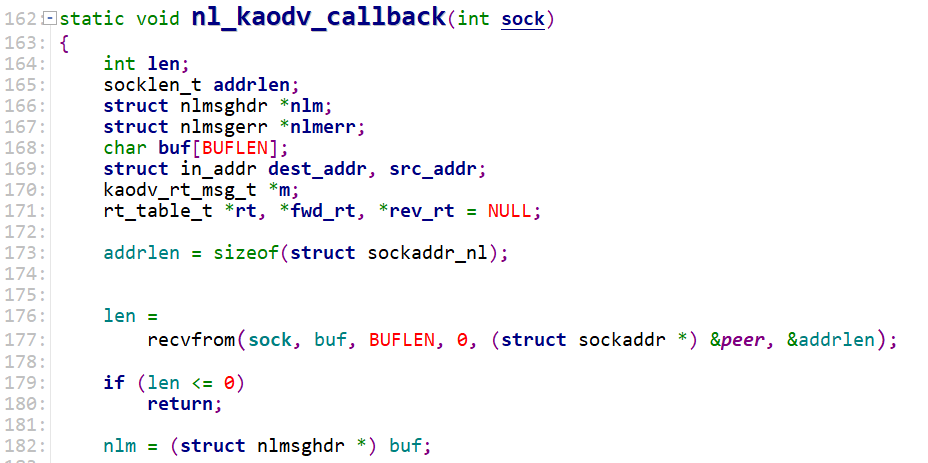


函数的后半部分

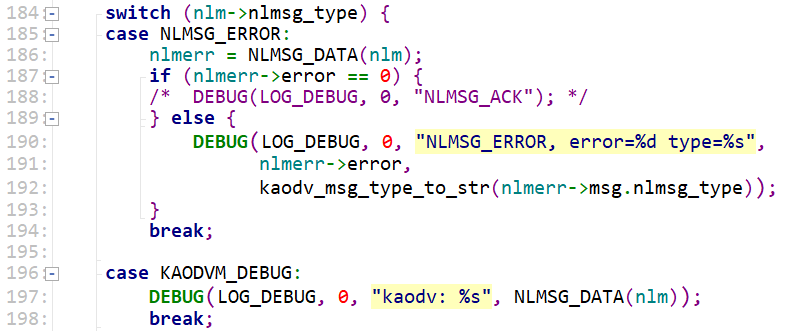
115~117 没有匹配的回调函数

121~152 和前面操作相同，也是新建一个套接字，不过这个套接字是用来与内核传递路由消息的。

## static void nl\_kaodv\_callback(int sock)



176~177 从套接字接收内核发来的消息，存到buf中

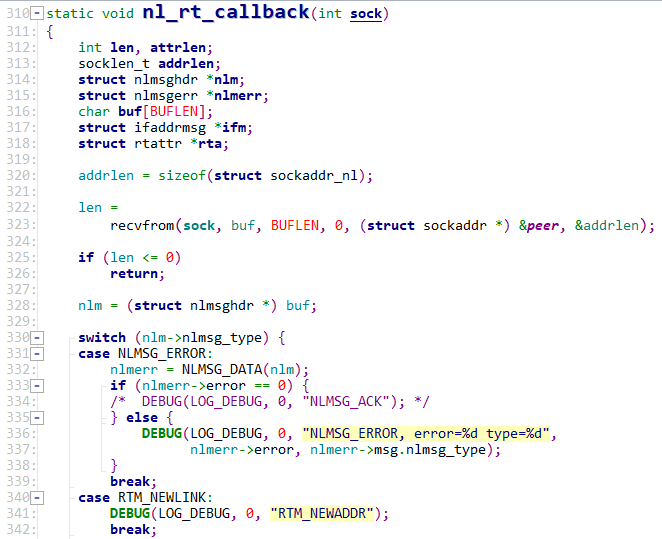


185~194 如果是ERROR消息，打印调试信息

196~198 如果是DEBUG信息，打印调试信息

之后的几个判断，也是同样，如果是TIMEOUT信息，调用route\_expire\_timeot函数，如果是路由请求消息，调用rreq\_route\_discovery函数等。

## static void nl\_rt\_callback(int sock)



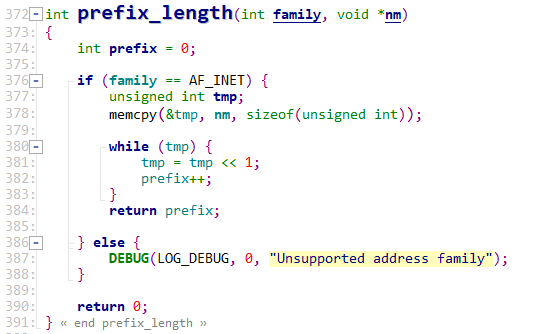
和nl\_kaodv\_callback一样，只是这个函数用来根据与内核进行路由消息传递的套接字的接收内容来决定调用什么函数。

331~339 如果是ERROR消息，打印调试信息

340~342 如果是RTM\_NEWLINK消息打印调试信息

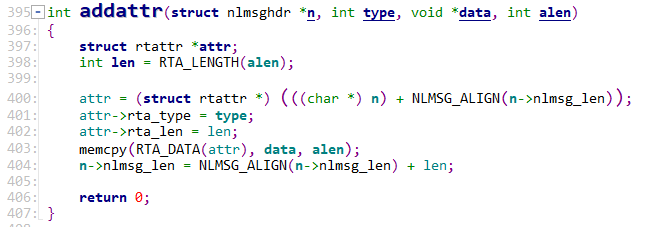
后半部分代码同上，如果是RTM\_NEWADDR、RTM\_NEWROUTE也是打印对应调试信息

## int prefix\_length(int family, void \*nm)



根据参数指定的地址族和IP地址计算出子网掩码长度即前缀长度

## int addattr(struct nlmsghdr \*n, int type, void \*data, int alen)



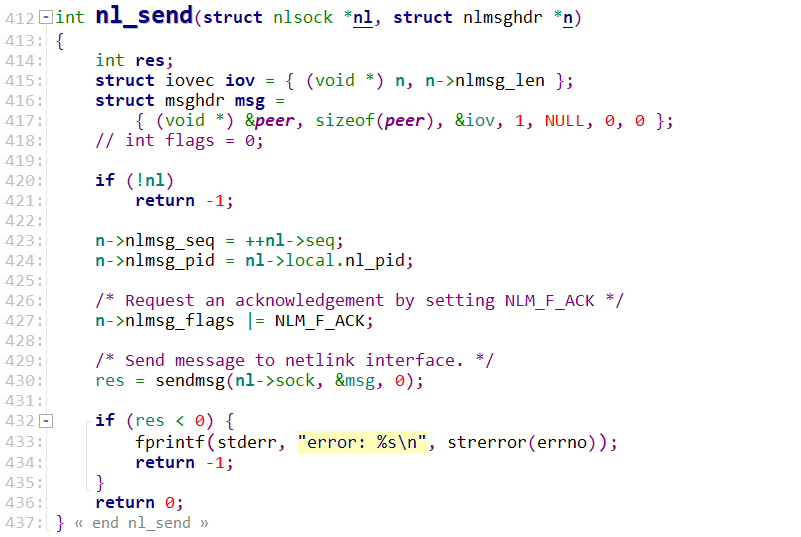
用于更新消息，增加一个属性

397 声明一个消息属性的变量attr

398 原长度更新为加上attr头部长度的总长

400~404 用于对齐和属性初始化。其中宏 NLMSG\_ALIGN (len) 用于得到不小于 len 且字节对齐的最小数值。

## int nl\_send(struct nlsock \*nl, struct nlmsghdr \*n)



把 netlink 消息嵌入到一个 iovec 结构体中然后用这个结构体构建msghdr消息，并将该消息发送给内核。

415~417 声明一个msghdr类型的变量，主要用于向socket发送消息或接收消息

## int nl\_kern\_route(int action, int flags, int family, int index, struct in\_addr \*dst, struct in\_addr \*gw, struct in\_addr \*nm, int metric)

用来添加、移除、更新内核路由表中表项的函数

## int nl\_send\_add\_route\_msg(struct in\_addr dest, struct in\_addr next\_hop,int metric, u\_int32\_t lifetime, int rt\_flags, int ifindex)

向内核发送一条netlink消息，消息内的areq消息头指示内核在表中添加一条路由信息

## int nl\_send\_no\_route\_found\_msg(struct in\_addr dest)

向内核发送一条netlink消息，将到达dest的路由信息设置为没有该条路由。

## int nl\_send\_del\_route\_msg(struct in\_addr dest, struct in\_addr next\_hop, int metric)

向内核发送消息将areq结构体中的路由m按照头部n的指示进行删除，随后调用nl\_kern\_route函数对路由表项进行删除

## int nl\_send\_conf\_msg(void)

通过套接字向内核发送类型为配置消息的areq消息。

# 参考部分，待删除

/\* 判断是否为合法的路由属性 \*/

8 #define RTA\_OK(rta,len) ((len) >= (int)sizeof(struct rtattr) && \

9 (rta)->rta\_len >= sizeof(struct rtattr) && \

10 (rta)->rta\_len <= (len))

11

12 /\* 获取下一个rtattr的首地址\*/

13 #define RTA\_NEXT(rta,attrlen) ((attrlen) -= RTA\_ALIGN((rta)->rta\_len), \

14 (struct rtattr\*)(((char\*)(rta)) + RTA\_ALIGN((rta)->rta\_len)))

15

16 /\* 返回加上 rtattr header的总长度 \*/

17 #define RTA\_LENGTH(len) (RTA\_ALIGN(sizeof(struct rtattr)) + (len))

18

19 /\* 返回数据对齐的最小值 \*/

20 #define RTA\_SPACE(len) RTA\_ALIGN(RTA\_LENGTH(len))

21

22 /\* 返回属性数据部分首地址 \*/

23 #define RTA\_DATA(rta) ((void\*)(((char\*)(rta)) + RTA\_LENGTH(0)))

24

25 /\*返回属性数据部分的长度 \*/

26 #define RTA\_PAYLOAD(rta) ((int)((rta)->rta\_len) - RTA\_LENGTH(0))

#include <sys/socket.h>

struct iovec { /\* Scatter/gather array items \*/

void \*iov\_base; /\* Starting address \*/

size\_t iov\_len; /\* Number of bytes to transfer \*/

};

struct msghdr {

void \*msg\_name; /\* optional address \*/

socklen\_t msg\_namelen; /\* size of address \*/

struct iovec \*msg\_iov; /\* scatter/gather array \*/

size\_t msg\_iovlen; /\* # elements in msg\_iov \*/

void \*msg\_control; /\* ancillary data, see below \*/

size\_t msg\_controllen; /\* ancillary data buffer len \*/

int msg\_flags; /\* flags on received message \*/

};

这里 msg\_control 指针: points to a buffer for other protocol control-related messages or miscellaneous ancillary data（指向与协议控制相关的消息或者辅助数据）. 而 msg\_controllen 为 msg\_control 所指向的这块缓冲的长度。

msg\_control 是一个 struct cmsghdr 结构，下面我们会介绍。

struct sockaddr\_nl

{

sa\_family\_t nl\_family; /\*该字段总是为AF\_NETLINK \*/

unsigned short nl\_pad; /\* 目前未用到，填充为0\*/

\_\_u32 nl\_pid; /\* process pid \*/

\_\_u32 nl\_groups; /\* multicast groups mask \*/

};

struct nlmsghdr//自定义消息首部，它仅包含了 netlink 的消息首部

{

\_\_u32 nlmsg\_len; /\* Length of message including header \*/

\_\_u16 nlmsg\_type; /\* Message content \*/

\_\_u16 nlmsg\_flags; /\* Additional flags \*/

\_\_u32 nlmsg\_seq; /\* Sequence number \*/

\_\_u32 nlmsg\_pid; /\* Sending process PID \*/

};