Netlink

关于 Netlink 的介绍，请参见 http://user.qzone.qq.com/944350535/blog/1403417856

下面介绍 Daoli Netlink 的实现。

为了支持多种类型数据的传递，因此使用一个字节作为服务类型，不同的服务类型传递不同类型的数据。因此，理论上最大支持 256 种服务类型。
实现方法：把传递的数据的第一个字节解析成服务类型，服务类型后面的数据（即从第二个字节开始）才是真正要传递的数据。API 接口会对此进行封装，使用者无需关心底层细节。

Netlink 用于内核空间与用户空间的通信，因此要涉及到内核空间和用户空间。

内核空间：

在模块启动时，首先要执行 **test_netlink_init**()函数，以便初始化 NETLINK；在模块退出时，要执行 **test_netlink_exit()**函数，以便清理 NETLINK。

接收：

要使用某种类型的服务（即接收用户空间下发的信息），首先要注册该服务类型。注册方法如下：

```
void register_service_handler(nl_recv_msg_t handler, __u8 type);
```

其中 type 是服务类型，handler 是一个函数指针，表示与 type 这种服务类型对应的的处理函数，其类型如下：

```
typedef void (*nl_recv_msg_t)(struct sk_buff *skb, struct nlmsghdr *nlh, void *data, size_t size);
```

其中，skb 是缓冲区，nlh 是从 skb 中解析出来的 NETLINK 消息头，data 是用户空间下发下来的真正数据（根据 NETLINK 消息头，从 skb 中解析出来），size 是数据的大小（以字节为单位）。一般来说，只需要使用 data 和 size 两个参数即可。
注：当用户空间下发某个服务类型的消息时，内核会自动调用上述注册的、相应的处理函数。

发送：

发送分为单播和广播。

```
int unicast_to_pid(void *data, size_t size, __u32 pid);
int unicast(void *data, size_t size);
int broadcast_to_group(void *data, size_t size, __u32 group);
int broadcast(void *data, size_t size);
int unicast_service_to_pid(void *data, size_t size, __u8 type, __u32 pid);
int unicast_service(void *data, size_t size, __u8 type);
int broadcast_servie_to_group(void *data, size_t size, __u8 type, __u32 group);
int broadcast_service(void *data, size_t size, __u8 type);
```

以 unicast 开头的是单播，以 broadcast 开头的是广播（或多播）。
上述接口的意思是:单播/广播 type 类型的、size 大小的数据 data 给 PortID 为 pid、组为 group 的用户空间 Netlink Socket。
如果没有 type 参数，则按默认类型 DEFAULT_SEND_TYPE（即 0）;
如果没有 pid 参数，则按默认 PortID DEFAULT_DEST_PORTID（即 1）;
如果没有 group 参数，则按默认组 DEFAULT_DEST_GROUP（即 1）。
如果成功，返回 0；否则返回-1。

用户空间：

已被封装成 Python 模块，可以使用 Python 接口直接调用。支持类和函数调用。

函数调用：

create(pid=DEFAULT_PID, group=DEFAULT_GROUP, protocol=NETLINK_PROTOCOL)
功能：创建一个 Netlink Socket。
参数：pid 是 Socket 要绑定的 PortID 号，默认是 1。如果是要与内核通信，请使用默认值。
group 是 Socket 要监听的多播组，默认是 1。如果为 0，表示不监听任何多播组。
注：用户空间要想监听多播组，必须拥有管理员权限（并不一定是 root）。
protocol 一般不要使用，除非它与内核注册的 NETLINK PROTOCOL 不一致。
返回值：返回一个与 Socket 关联的文件描述符；如果出错，返回 -1、-2 或-3。

recv(fd, type=DEFAULT_RECV_TYPE)
功能：接收一个 Netlink 消息。
参数：fd 是 Netlink Socket 的文件描述符（即 create 的返回值）。
type 是要接收的服务类型（一个整数），默认是 0。
返回值：返回一个元组，即(data, size, type, flags, seq, pid)，其中，
data 是内核空间发送过来的内容实体（字节类型字符串）;
size 是 data 数据的长度（以字节为单位）;
type、flags、seq、pid 等信息分别是 struct nlmsghdr 结构体中 nlmsg_type、nlmsg_flags、nlmsg_seq、nlmsg_pid 等字段的值。
其实，只需要关心 data 和 size 两者信息即可。
注：如果 data 是结构体数据，则需要使用 struct 模块进行解析成相应的数据。
如果出错，则返回 None。

send(fd, data, size, type=0, pid=0, group=0)
功能：发送一个 Netlink 消息（接收者是 PortID 为 pid 并监听 group 组的所有 Netlink Socket）。
参数：fd 是 Socket 文件描述符（create 的返回值），
data 是要发送的数据（必须是字节型），
size 是 data 的长度（以字节为单位），
pid 是接收消息的 Socket 的 PortID（默认为 0，即内核 Socket），
group 是多组播（默认是 0，即不发送给任何多播组）。
type 是数据的服务类型，即将数据当作此服务类型发送。
返回值：返回实际发送的数据的字节数；如果出错，则返回一个负数：
-1 表明发送失败， -2 表明调用出错（有可能参数不对）。

close(fd)
功能：关闭一个 Netlink Socket。
参数：fd 是 create 的返回值。
返回值：永远返回 None。

类 **Netlink**

class **Netlink**(object):
def **__init__**(self, pid=1, group=1, dst_pid=0, dst_group=0, protocol=30):
同 create 函数。如果成功，返回一个 Netlink 对象；否则，抛出一个 Exception 异常。

def **recv**(self, type=DEFAULT_RECV_TYPE):
同 recv 函数。

def **send**(self, data, size, type=DEFAULT_SEND_TYPE, pid=None, group=None):
同 send 函数。如果 pid 或 group 为 None，则使用 dst_pid 或 dst_group 参数。

def **close**(self):
同 close 函数。