Ring 3 DLL

结构及说明;

```
//过滤规则 结构
typedef struct filter_condition {
    //此结构可随着功能的增加自定义，和驱动的相同结构一起改

    UCHAR S_Or_R;//在发送时过滤还是收到时过滤 0：发送 ，1：收到
    UCHAR S_Or_D;//是过滤源还是目的 0：源，1：目的
    UCHAR V4_Or_V6;//是过滤ipv4还是ipv6  0:ipv4  1:ipv6
    union
    {
        UCHAR ipv4[4];
        UCHAR ipv6[16];
    }IP;
    USHORT port;
    int flag;//标识号，删除时要用
    //UCHAR flag[20];//对应使用的过滤函数：1使用，0不使用，预留20个过滤条件函数的位置
    long long B;//占位用的，无实际意义
    long long A;//占位用的，无实际意义

}Filter_Condition, * PFilter_Condition;
```

```
//会话结构
typedef struct record {
    UCHAR v4_or_v6;
    union
    {
        UCHAR ipv4[4];
        UCHAR ipv6[16];
    }IP;
    long long s;//占位用的，无实际意义
}Record, * PRecord;
```

输出函数:

```
//启动RING 0 RING 3 交互
int WINAPI init();
```

```
//开启过滤
int WINAPI start();
```

```
//停止过滤
int WINAPI stop();
```

```
//添加ipv4过滤规则，输入规则标识号，ip，端口，在发送时还是接收时拦截，此ip的源ip还是目的ip
int WINAPI ipv4_new_rule(int flag,int ip[],  int _port,int S_Or_R,int S_Or_D);
```

```
//添加ipv6过滤规则，输入规则标识号，ip，端口，在发送时还是接收时拦截，此ip的源ip还是目的ip
int WINAPI ipv6_new_rule(int flag, int ip[], int _port, int S_Or_R, int S_Or_D);
```

```
//添加进程过滤规则，输入规则标识号，进程PID，在发送时还是接收时拦截
int WINAPI process_new_rule(int flag,int PID,int S_Or_R);
```

```
//删除规则，输入此规则的标识号
int WINAPI dele(int flag);
```

```
//删除所有规则
int WINAPI dele_all();
```

```
//获取恶意会话ip的数量
int WINAPI get_m_ip_num();
```

```
//获取第一个恶意会话ip，输入一个整数型数组，数组长16
int WINAPI get_first_m_ip(OUT int *out);
```

```
//获取下一个恶意会话ip，输入一个整数型数组，数组长16
int WINAPI get_next_m_ip(OUT int* out);
```

说明：

在进行过滤规则的增加删除前，先停止过滤，否则会增加、删除失败。

关于进程过滤

有小BUG，可能会引起程序异常结束。

原理是通过进程PID获取该进程此时占用的TCP、UDP端口，然后作为拦截规则传入内核。

换言之如果 该进程另外再申请用TCP、UDP端口的话，是无法拦截的。

给出以下多种解决方法（包括但不限于）：

1.通过HOOK需要拦截的进程里的相关函数实现进程每申请一个TCP、UDP端口就通知内核增加规则。

2.通过HOOK内核相关函数实现同上功能。

3.通过PID获取端口后，再通过端口获取 远程进程的IP地址，再以该IP作为拦截规则。

4.通过循环或一个线程 读取判断需要拦截的进程 是否申请了新的TCP、UDP端口，若有新的，则增加拦截规则。

个人比较推荐使用第三种。但是可能会拦截到其他同样访问该ip的进程。

第一第二种我认为更加稳妥，但是可能需要考虑进程的反调试或是系统PG的问题。