基于 VxWorks 的以太网驱动程序设计

高　云　刘广武

（中国航天科工集团三院 8357 所）

摘要：本文论述了 VxWorks 的以太网驱动程序（END）的组织结构和编写方法，叙述了其与操作系统的接口和与网络协议层的通信机理。

关键词：VxWorks　MUX　END

一、引言

VxWorks 操作系统由于其良好的抗恶劣环境的性能和卓越的实时性，在军事和航天领域得到了广泛的应用。目前，VxWorks 已成为事实上的工业标准和军用标准。

VxWorks 提供了常用的设备驱动程序，用户只需在配置文件中指定设备的类型,规格和重要的参数即可,不用编写驱动程序,但遇到下面两种特殊情况时,用户需要自己编写驱动程序。

(1)用户使用的设备比较新，操作系统不支持。

(2)用户想在底层实现自己的特殊要求，如双网卡冗余、大容量吞吐、数据备份等。此时用户必须在驱动程序中加上实现所需功能的代码。

VxWorks 支持的驱动函数由两种（如图 1）：一种是标准的 BSD4 驱动程序。它将驱动程序和协议紧密的联系在一起，不利于多协议的支持。VxWorks 不推荐使用 BSD4 驱动程序。另一个标准是 VxWorks 专有的 END(enhanced network driver)驱动程序。它通过一个称为 MUX 的薄层,将驱动程序和协议层隔离开来，达到驱动程序独立于具体协议栈的目的，从而实现多协议的支持。

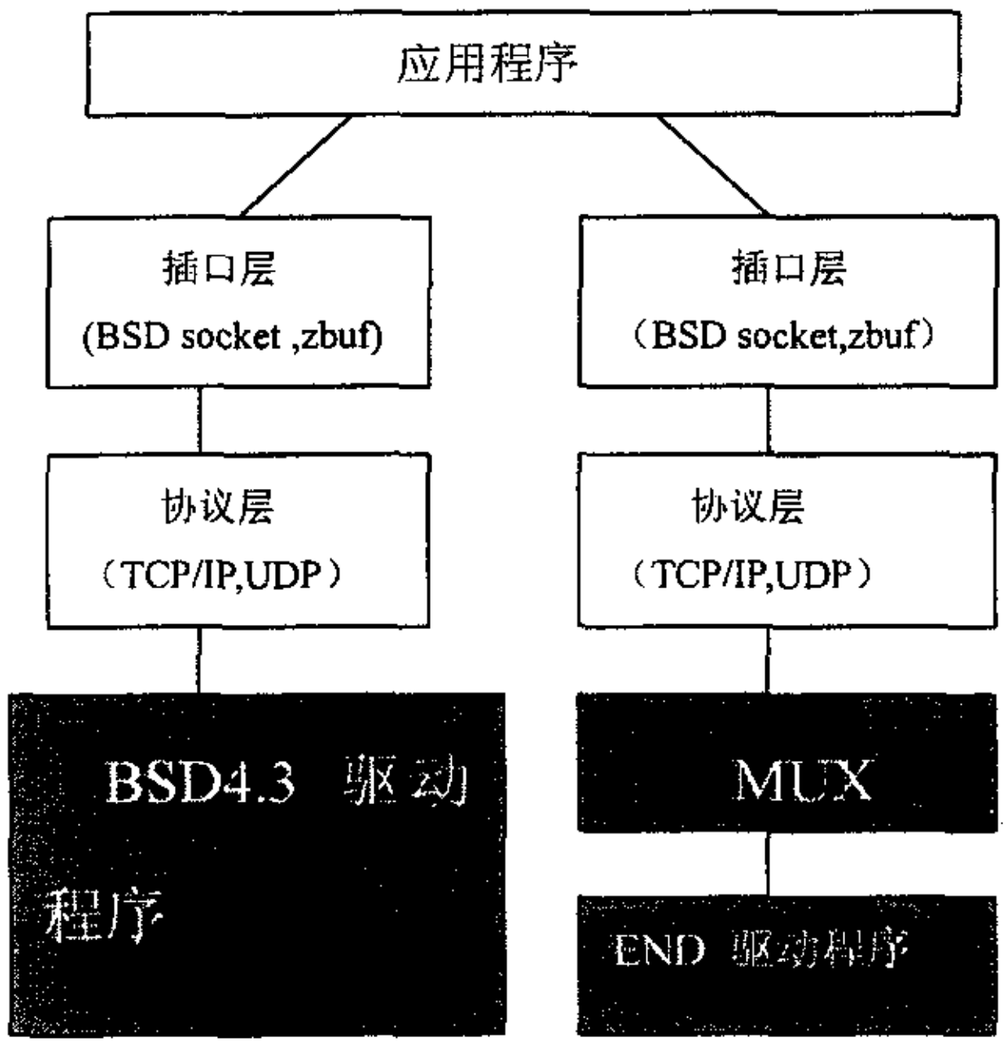

图 1 驱动程序结构关系图

下面我们将对 END 驱动程序和其相关问题进行详细讨论。

二、数据链路层和网络层之间的一个接口——MUX 层

VxWorks 提供的 MUX 接口分割开了数据链路层和网络层，这样就使得数据链路层和网络层更加独立。为了使用位于数据链路层的驱动函数，网络协议必须调用 MUX 接口层所提供的的 MUX 函数;同样地，当位于数据链路层的驱动函数想要访问网络层协议时，它也必须调用 MUX 函数。所以说 MUX 接口层即分离了数据链路层和网络层，同时又在它们之间起到桥梁作用。由于协议和驱动函数不直接访问彼此，彼此不需要太多的了解，这使得加载新的驱动程序或协议变得更加容易。

2.1 MUX 层提供的 API 函数

无论网络协议层还是网络接口驱动函数都可以把 MUX 看作应用程序接口(API)。此接口使得与其相连的网络协议和网络驱动函数只需在此接口的基础上编写即可，增强了彼此的独立性和编程更加容易。MUX 层、网络协议层和网络接口驱动之间的调用关系如图 2 所示。

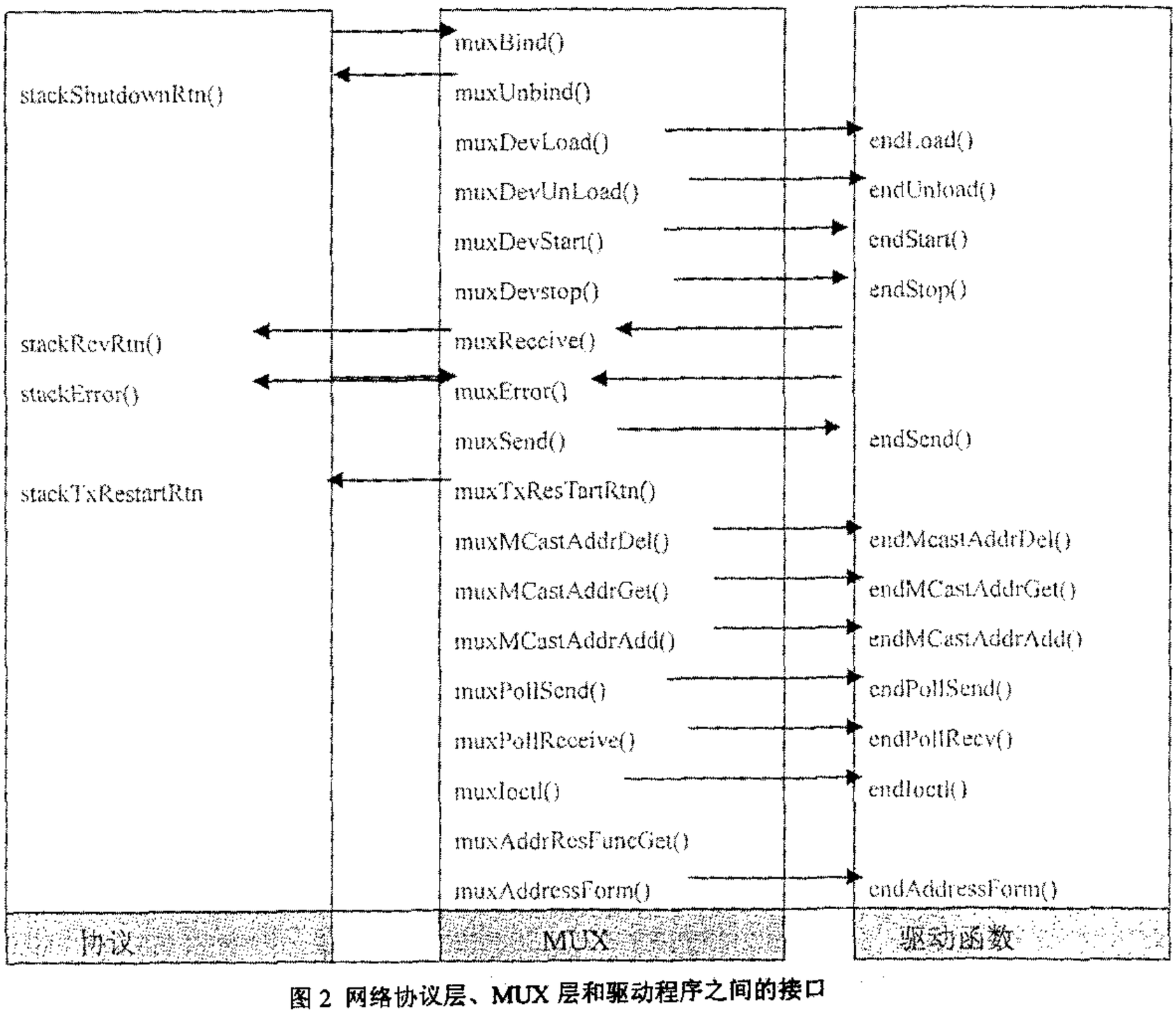

图 2 网络协议层、MUX 层和驱动程序之间的接口

下表为 MUX 所提供的 API 函数的简单说明。

muxDevLoad()	通过调用 endload()装载设备进入 MUX 层
muxDevStart()	此函数将通过调用驱动函数的 endStart() 函数激活网络接口
muxDevStop()	通过调用 endstop()函数停止一个网络设备
muxDevUnload()	此函数从 MUX 中卸载一个设备,捆绑在此设备上的协议将与此设备断开
muxbind()	将协议与指定的驱动函数相捆绑
muxunbind()	将协议与指定的驱动函数相分离
muxsend()	通过网络接口发送数据报
muxpollsend()	当驱动程序处于查询模式而非中断模式时使用此函数来发送数据报
muxpollreceive()	网络协议使用此函数以查询方式接收从设备输入的数据报
muxioctl()	向 MUX 层发送控制命令
muxMCastAddrAdd()	向设备的广播地指表中增加一个广播地址
muxMCastAddrDel()	从设备的广播地址表中删除一个广播地址
muxMCastAddrGet()	从设备的广播地址表中获得广播地址
muxAddressForm()	网络协议将调用此函数构造数据链路层的帧,此函数将形成一个加上了链路层头的新数据报
muxPacketDataGet()	从数据报中获得数据
muxPacketAddrGet()	从数据报中获得地址信息
endFindByName()	以设备名和设备单元号位参数，找到与此参数相匹配的 END 设备
muxDevExists()	这个函数将检测是否一个特定的设备在 MUX 层中已经存在。
muxAddrRseFuncAdd()	为指定的接口类型和协议类型增加地址解析函数
muxAddrRseFuncDel()	从地址解析表中移走为此接口和协议类型所登记的地址解析函数
muxAddResFunGet()	得到一个指定接口和协议类型的地址解析函数

三、END 驱动程序的编写

3.1 数据结构 END_OBJ

在嵌入式系统中，为了保证实时性，一般为设备定义一个数据结构。在数据结构中存储了设备的一些重要的参数：如基地址、中断号、END_OBJ数据结构等。在调用endload（）函数进行网卡初始化时，对这个数据结构进行操作，将具体的设备参数填入此设备的数据结构中。数据结构的定义随网卡的不同而各不相同，但无论是用那种网卡在各种网卡的数据结构中必然包含一个END_OBJ结构，此结构是END驱动程序的核心数据结构，存储着接口的重要信息，驱动函数必须分配和初始化这个结构。在VxWorks中这个结构定义在target/h/end.h文件中。此结构中的一些成员由MUX层来管理，驱动函数负责设置和管理其它的成员。

3.2 END 驱动程序为MUX提供的接口

从图2我们就可以看到END驱动程序为MUX层提供了丰富的接口函数。通过这些接口函数实现了MUX层与驱动函数之间的通信，从而将网络协议与驱动程序连通起来。下表中为END驱动程序中用于同MUX层通信的驱动函数的列表。

endload()	装载一个设备进入MUX层
endunload()	从MUX层释放一个设备或者设备上的一个端口
endsend()	从MUX层接收数据和发送他到物理层
endMCastAddrAdd()	在地址列表中增加一个广播地址
endMCastAddrDel()	从地址列表中移出一个广播地址
endMCastAddrGet()	获得广播地址列表
endpollsend()	通过查询方式来发送帧
endpollReceive()	通过查询方式来接受帧
endStart()	连接设备中断和激活接口
endStop()	停止或使一个网络设备无效
endAddressForm()	在数据包中增加地址信息
endAddrGet()	从数据包中获得地址信息
endPacketDataGet()	分离数据包中的地址信息和数据
endIoctl()	支持各种ioctl命令

在这里我们使用了 end 作为前缀在实际中前缀由具体的驱动函数来确定。

3.3 END驱动程序中典型构成部分的分析

从对END函数的列表中我们可以看到要实现END驱动函数的全部功能，驱动程序将变得十分复杂，这里我们只对驱动程序中几个重要的构成部分进行讨论，这几个部分是驱动程序中必不可少的通过它们可以构建最基本的驱动程序。它们分别为：网卡初始化、中断处理、内存管理，数据报的接收和发送。

3.3.1 网卡初始化

由于嵌入式系统受到工作环境的限制，要用很少的代码实现其强大的功能，就不能象windows系统一样自动诊测设备参数，自动给设备分配资源。因为那样将耗费大量的代码空间。所以在VxWorks中的bsp提供了config.h文件，通过它用户可以指定设备的规范、中断、基地址等重要参数，设备初始化函数则根据用户在config.h文件中的配置来初始化设备。初始化过程主要是通过初始化函数和参数解析函数来实现的，初始化过程主要如下。

a. 为包含END_OBJ结构的网卡数据结构分配存储空间，调用参数解析函数(parse（）)对用户在config.h文件内定义的网卡参数进行解析，以便对此网卡数据结构中的一些参数进行初始化。用户的配置参数存放在config.h配置文件中，并以字符串的形式传递给初始化函数（endload()）,这种初始化字符串包含的重要参数为设备单元号、基地址、中断向量号、中断优先级、8字节的访问模式、从PROM中获得的以太网地址、内存偏移量等。

b. 对网卡数据结构中的END_OBJ结构进行初始化。通过对其初始化完成本设备函数表的装入和管理信息库的初始化。

c. 网卡内存初始化，对网卡内存的初始化是为了适应协议和操作系统对内存控制的机制。一般做法是为网卡对象分配一块受协议栈或操作系统控制的内存区域，此内存区域被分为mBlk结构池,clBlk结构池和cluster(族)池。数据存储在cluster中，并通过mBlk-clBlk-cluster链的形式来引用。

d. 通过对END_OBJ结构中的flags的设置表明此设备已准备好。

3.3.2内存管理

VxWorks 提供了有效的内存分配和管理方法。这主要体现在两个方面。第一 VxWorks 提供了统一的数据存储结构（mBlk-clBlk-cluster 链）这使得我们可以通过 mBlk 结构来访问 cluster 中的数据，通过对 mBlk 结构的使用，使我们避免了协议层之间的大量的数据拷贝，而只是在不同的协议层中创建 mBlk 链来索引到所需的数据，而 clBlk 结构中包含着指向相同 clBlk-clueter 结构的 mBlk 的个数。第二 VxWorks 提供了丰富的系统函数用来管理内存的占用和释放。图 3 为不同的 mBlk 分享同样的 cluster 中数据的 mBlk-clBlk-cluster 链的形式。

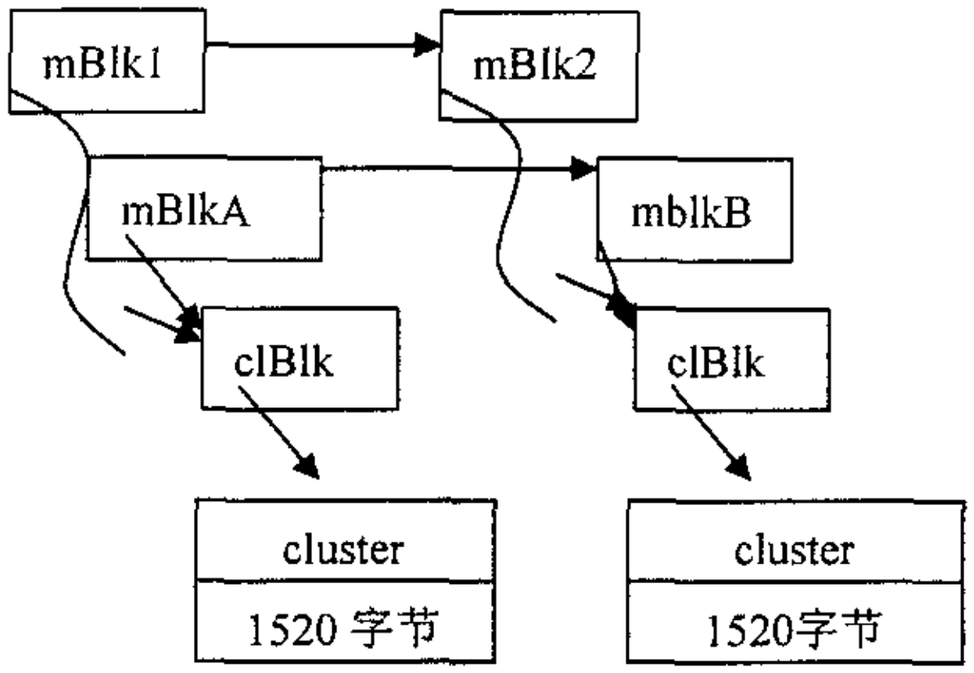

图 3 mBlk,clBlk,cluster 三者之间的关系

驱动程序通过调用内存初始化函数（netpoolinit()）在内存中分配了一个内存池，此内存池中包含几个子池，对于驱动程序来说，它包含一个 mBlk 结构池，一个 clBlk 结构池，和一个有统一族尺寸的 cluster 池，此族尺寸为与设备的 MTU(最大传输单元)相近的值。对于以太网驱动程序通常的族尺寸为 1520 字节。如图 4 所示。

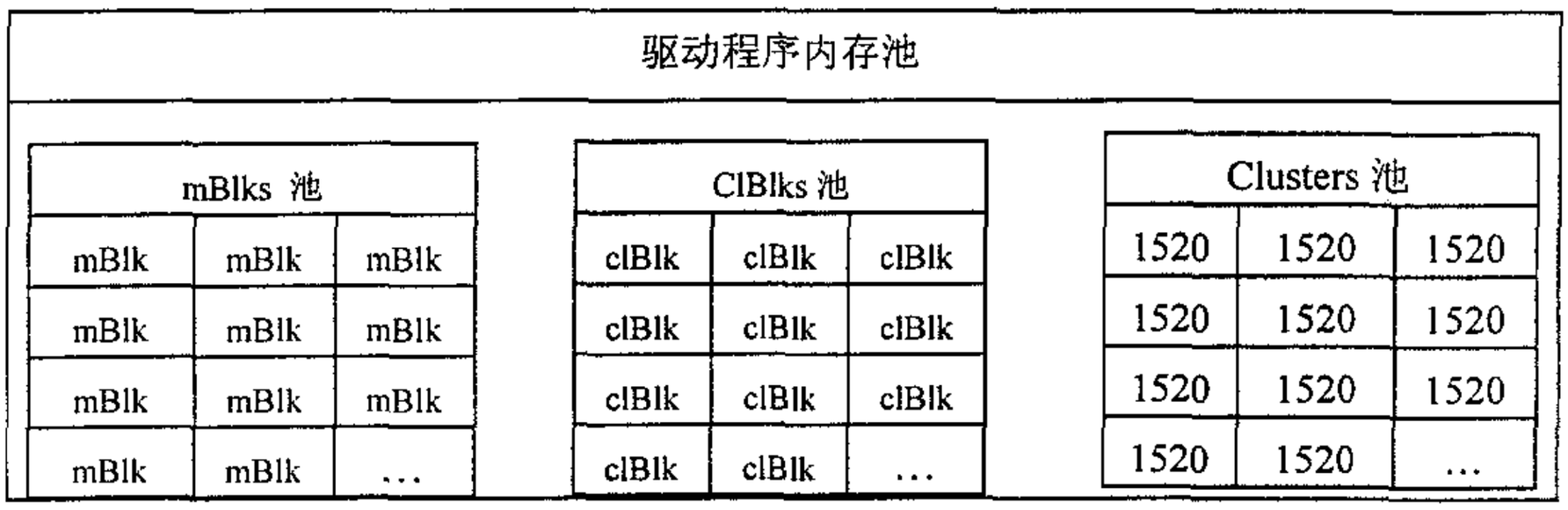

图 4 驱动程序初始化的内存池

netBufLib 中提供内存管理函数如下表所示。

netBufLibInit()	初始化 netBufLib
netpoolinit()	初始化一个由 netBufLib 管理的内存池
netpoolinit()	删除一个内存池
netMblkFree()	释放 mBlk 结构到内存中
netClBlkFree()	释放 clBlk—cluster 结构到内存中
netClFree()	释放 cluster 到内存中

netMblkClFree()	释放 *mBlk-clBlk-cluster* 结构
netMblkClChainFree()	释放存有数据的 mBlk 链表
netMblkGet()	从内存池中取 mblk 结构
netClBlkGet()	从内存池中取 clBlk 结构
netClusterGet()	从内存中得到 cluster
netMblkClGet()	从内存中得到 clBlk-cluster 并联结它到指定的 mBlk
netTupleGet()	从内存中得到 mBlk-clBlk-cluster 结构
netClBlkJoin()	将 clBlk 与存有数据的 cluster 联结起来
netMblkClJoin()	将 mBlk 与 clBlk-cluster 结构联接起来
netClPoolIdGet()	从与指定的缓冲区尺寸相匹配的 *cluster* 池中获取 CL_POOL_ID 值
netMblkTobufCopy	拷贝 mBlk 链中数据到缓冲区中
netMblkDup()	对 mBlk-clBlk-cluster 结构中的 mBlk 进行复制
netMblkChainDup()	复制 mBlk 链

3.3.3 中断处理

网卡的中断处理函数用于当网卡以中断方式工作时，如果设备产生中断，进行中断响应，进而根据中断状态寄存器的值来处理报文接收，出错处理等不同的中断。中断函数一般分为两部分。第一部分是一些耗时较小的响应中断的函数（如 溢出错误处理、报文接收错误处理、报文发送错误处理等中断例程）。这些中断例程通过bsp中提供的系统函数sysintconnet()挂接到系统的中断结构上，一般在muxDevstart()完成此操作。第二部分是耗费时间的中断处理例程（如数据报处理例程）。在实时系统中驱动程序应避免对这些数据报处理例程直接调用，以减少中断关闭的时间，而是通过调用系统函数netjobAdd()将数据处理函数放在netjob任务的网络队列中，通过系统任务tNetTask来处理。

3.3.4 数据报的接收

设备直接将接收到的数据报放入内存池预先分配的cluster中和产生一个中断。如果设备不能完成上述功能，end驱动函数完成将数据从buffer到cluster中的拷贝。数据被放cluster 中后，驱动程序将通过对netBufLib中函数的调用来完成mBlk/clBlk/cluster链的创建，从而为数据在网络协议各层之间的传递做好准备，创建此结构链一般需要以下四步：

a. 调用函数netClBlkGet()从内存池中取clBlk结构。

b. 调用函数netClBlJoin()将clBlk与存有数据的cluster联结起来。

c. 调用函数netMblkGet()从内存池中取mBlk结构.

d. 调用函数netMblkClJoin()将mBlk与clBlk-cluster结构连接起来。

为了向上层协议栈传递此结构链，end还将调用muxLib中所提供的函数，具体的传递过程一般分为以下两步：

a. end 调用end_obj结构中的成员receiveRtn(muxDevLoad()已将receiveRtn成员初始化位函数muxReceive().

b. muxReceive()调用协议层的入口函数stackRcvRtn.

数据报的接收过程如图5所示，

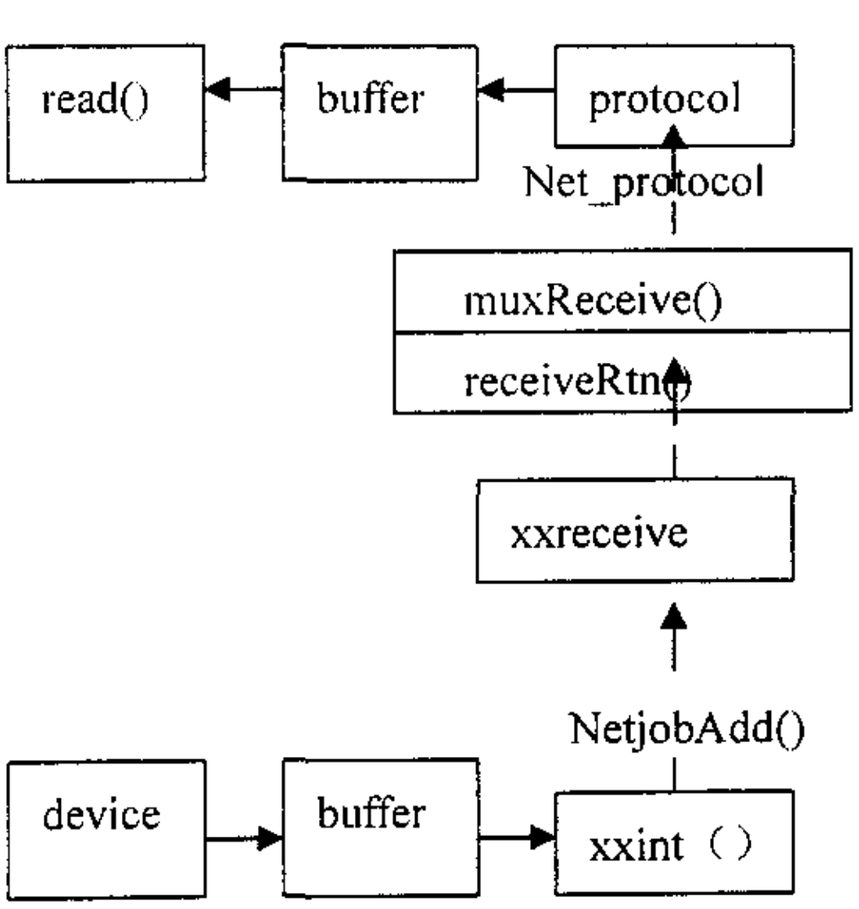

图5 数据报的接收过程

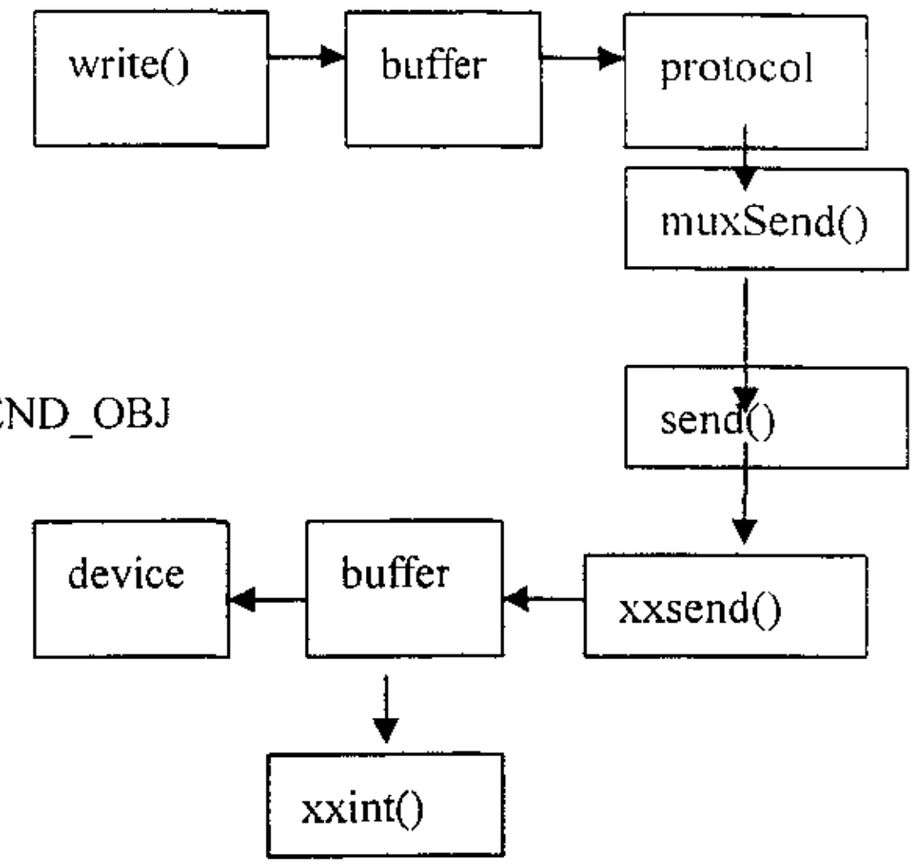

图6 数据报的发送过程

当一个数据报到来时会触发一个中断，中断服务程序（xxint）将通过调用netjobAdd()项任务队列添加一个网络任务，此网络任务为数据报接收函数（xxreceive），它通过系统任务tNettask来调用。然后接收函数调用mux的接口函数muxReceive，而muxReceive又调用协议层提供的接口函数stackRcvRtn将数据报传递到协议层，最终数据将通过协议层到达应用程序的缓冲区中，应用程序通过read()函数对其读取。从图5中我们可以看到，数据报经过物理层到达数据链路层，然后再通过mux层到达网络层，在通过TCP协议层到达应用层，完成了数据报接收的全过程。

3.3.5 数据报的发送

数据报的发送基本上是数据报接收的反过程，如图6所示。应用程序通过write()函数调用，将要发送的数据放入应用程序数据缓冲区（buffer）中。网络协议负责将buffer中的数据放入为其分配的内存池中，并以mblk-clblk-cluster链的形式来存储,这样实现了再往下层协议传递数据报时，传递的只是指向此数据链结构的指针，而代替了数据在各层协议之间的拷贝。当由数据要发送时，网络协议层通过其与mux层的接口调用muxsend()函数，而muxsend()函数又通过调用xxend()函数负责将利用指针传递来的数据报送到发送FIFO队列中，然后起动网卡设备的发送功能，发送完后将随之产生中断信号,调用中断服务程序，清除设备缓冲区。

四、结束语

本文对END驱动程序所涉及的一些问题进行了总结，但还有更多复杂的问题本文并未进行讨论。编写出好的END驱动程序应体现在它的实时性和稳定性上，而不是它的功能如何强大，所以编写END驱动程序应结合具体的硬件设备和瞄准所要完成的功能来编写。

参考文献

[1] WindRiver Systems Incorparated. Network Protocol ToolKit user's guid.

[2] WindRiver Systems Incorparated. Vxworks Networks Programmer's guid.

[3] WindRiver Systems Incorparated. Tornado Tainning Workshop.