1. 概述

为了实现应用层程序的平台无关性，操作系统为应用层提供了一套标准的接口函数，这些接口函数在所有的平台上都保持一致，只是随着平台的变化，底层驱动或接近驱动部分操作系统中间层可能会随着调整。这样可以使用户程序独立于具体的硬件平台，增加了应用层开发的效率，避免了重复编码。通用操作系统GPOS(General Purpose Operating System)比如Unix,Linux，将这套提供给应用层的标准接口函数从操作系统中独立出来，专门以标准库的形式存在，增加了应用程序的平台无关性，平台之间的差别完全被操作系统屏蔽。

和Unix,Linux类似，VxWorks也对应用层提供了一套标准文件操作接口函数，实际上与GPOS提供接口类似，我们将其称作为标准I/O 库，VxWorks 下由ioLib.c 文件提供。ioLib.c 文件提供如下标准接口函数：creat、open、unlink、remove、close、rename、read、write、ioctl、lseek、readv、writev 等。VxWorks 操作系统区别于GPOS的一个很大不同点是：VxWorks 不区分用户态和内核态，用户层程序可以直接对内核函数进行调用，而无须使用陷阱指令之类的机制，以及存在使用权限上的限制。因此VxWorks 提供给应用层的接口无须通过外围库的方式，而是直接以内核文件的形式提供。用户程序可以直接使用ioLib.c 文件定义的如上这些函数，这些函数名称与GPOS标准库下的函数名一致，是VxWorks 对标准库的模拟。

VxWorks的I/O系统给应用程序提供了简单、统一、与下列设备无关的接口：

1. 面向终端的字符设备或者通信线路设备；
2. 随机访问块设备，例如硬盘；
3. 虚拟设备，例如管道和套接字；
4. 用于监控和控制的数据或者模拟I/O设备；
5. 访问远程设备的网络设备。
6. VxWorks I/O框架

VxWorks的I/O系统和Unix或者Linux的I/O系统的不同之处在于，响应用户请求的工作分布在与I/O系统无关的设备和设备驱动本身之间。

在GPOS中，设备驱动提供某些底层的I/O例程用于从字符串定位的设备中读取或者写入字符序列。基于字符设备的高级别通信协议协议例程，通常由I/O系统中独立于设备的部分实现。在驱动例程获得控制权前，用户请求严重依赖于I/O系统的服务。

尽管这种解决方案使得驱动实现起来较为容易，并且驱动的行为表现的尽可能的相似，但是它存在这种缺陷：即当驱动的开发者实现现存的I/O系统中不存在的协议时，将面临极大的困难。而在实时系统中，如果某些设备的吞吐量至关重要时，我们需要绕过标准的协议，或者这类设备本身就不适合标准的模型。

VxWorks系统中，在控制器权转到设备驱动程序之前，用户的请求进行尽可能少的处理。VxWorks I/O系统的角色更像是一个转接开光，负责将用户请求转接到合适的驱动例程上。每一个驱动都能够处理原始的用户请求，到最合适它的设备上。另外，驱动程序开发者也可以利用高级别的库例程来实现基于字符设备或者块设备的标准协议。因此，VxWorks的I/O系统具有两方面的优点：一方面使用尽可能少的使用驱动相关代码就可以为绝大多数设备写成标准的驱动程序，另一方面驱动程序开发者可以在合适的地方使用非标准的程序自主的处理用户请求。

设备一般有两种类型(这里暂时不考虑网络设备)：块设备或者字符设备。块设备用于存放文件系统，在块设备中数据以块的形式存放，块设备采用随机访问的方式，像硬盘或者软盘都属于块设备；不属于块设备范畴的设备被称为字符设备，向串行设备或者图形输入设备都属于字符设备。

VxWorks的I/O系统包含三个元素：驱动、设备和文件。接下来我们以字符设备为例，其中大部分的分析也适用于块设备。当然，块设备必须和vxWorks 文件系统进行交互，所以其在组织结构上和字符设备稍有不同。