每个 RAM 盘用次设备号来区分。一般情况下这些区域相互分开，但是在某些情况下，也不排除相互重叠的可能性。

#### 3.2.3.2 RAM 盘的软件

和我们的惯例一样，RAM 盘的驱动程序也需要实现前面设备驱动程序接口中提到的函数。我们将这些实现都放在 ramdisk.c 中，而与之相关的定义在 ramdisk.h 中。ramdisk.c 包含了对 RAM 盘操作的四个函数入口，它们是：name（返回一个设备名称）、read、write、geometry（返回硬件参数）。可以看到对于 RAM 盘来说，不仅函数实现少，而且实现的这几个函数基本上都属于没有什么操作的例程，这是因为 RAM 盘的操作并不复杂，其它函数是多余的。

在实现细节上要说明的一点是，缺省情况下，系统不自动建立 RAM 盘（这是因为嵌入式系统的 RAM 资源非常宝贵）。配置 RAM 盘的大小的工作放在系统配置文件中，要求用户提供一个最大和最小的 RAM 盘空间，系统会根据可以使用的内存数量寻找一块足够大的空间来作为 RAM 盘，如果能使用的内存不满足用户给出的最小值，则返回一个错误。这样实现的好处是：不仅能使用户的配置简化，而且不用重新编译内核就可以给出不同大小的 RAM 空间。

对于 RAM 盘的读写操作，简单到可以用 ANSI C 库函数 memcpy 来实现。而最后一个函数 geometry，是在一旦 RAM 盘被询问到有关磁道、柱面、扇区等物理特性的时候，用于模拟提供这些参数的一个函数，它的实现是一系列的赋值语句，然后 return。

### 3.2.4 空设备

空设备实际上是 RAM 盘的一种，它的功能是接受输入数据并把数据抛弃掉的一种设备。一般情况下没有太大的用处，但在使用 shell 程序的时候有特殊的意义，例如程序执行的结果不再需要了，或者不想让程序的结果输出到标准输出设备上（一般是显示器）时，就可以这样使用

```
cat readme.txt > NULL
```

RAM 盘的驱动程序对它采取一种高效的处理方式，将它的长度设置为 0，这

样就没有数据对它拷入或拷出。

## 3.3 块设备管理的主要数据结构和实现

### 3.3.1 设备管理模块的结构

设备管理模块实现对系统中所涉及到的块设备的管理，当然还要为以后可能出现的块设备留出可扩展的空间，参考 linux 的设备驱动程序实现，我们采用了图 3-7 的系统结构：

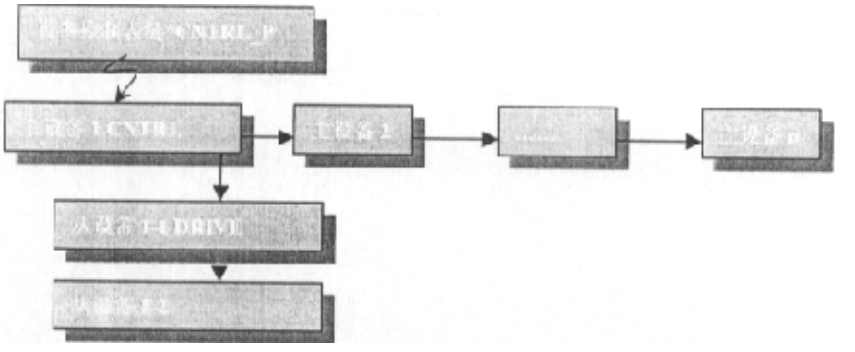

图 3-7 设备管理模块结构

图 3-7 中，采取了一种倒置的树状结构，由设备控制块的全局变量*CNTRL_P 控制整个设备表（该结构将在本小节后面作详细说明）。它的下一层结构是主设备表，主设备表的项数是在系统初启的时候由用户定义的系统所要支持的块设备数决定的，这意味着每次增加一种硬件，就需要重新编译一次系统，这和我们在§3.2.1 中说明的思想一致。

设备树的叶子节点是各个从设备，系统中主设备和从设备的界定是：如果设备使用了单独的中断，则使用相同中断的各种设备共用一个主设备号；如果没有使用中断，则是用同一种设备驱动程序的各种设备共用一个主设备号。上面的叙述看起来很令人费解，但是我们却认为它很实用，这使我们在理解几个硬盘都是用同一个主设备号的基础上，不难理解 RAM 盘和空设备也要使用同一个主设备号。

在图 3-7 中，标出了用于各个设备层次的数据结构的名称，本小节的后面部分就是对这些数据结构的详细描述。