Linux系统下设备驱动的安全端口分配 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/144/2021_2022_Linux_E7_B3_BB_E7_BB_c103_144163.htm 摘要： 编写设备驱动是一个具有挑战性和冒险性的工作。当设备通过init_mo dule函数登记时，设备的资源应当被分 配。一个主要的设备资源是I/O端口。作为动态连接的驱动程序，开发者应当小心将未被使用的I/O端口分配 给这些设备。 首先驱动程序应侦测这些端口是否被使用或释放。然后再为设备申请获取端口。当驱动模块被 从内核中移出时，端口应该被释放。这篇文章讨论了Linux设备驱动的安全端口分配的复杂性。 介绍 设备驱动开发者一个主要关心的问题是设备的资源分配。这些资源包括I/O端口，内存和中断。这篇文章试图解释I/O子系统的基本原理和资源分配的重要性，主要是I/O端口的资源处理。同时还将阐明如何侦测，申请和释放设备的端口地址。 基本的硬件元素，如端口，总线和设备控制器，构成了大量的不同的I/O设备。设备驱动向I/O子系统提供了 一个通用的设备存取界面，这非常类似于系统调用（systmem call）在应用程序和操作系统之间提供的标准界 面。现在有很多种类型的设备附属在电脑上，举例说来有： 存储设备，如磁盘，磁带，光驱和软驱； 人机交互设备，如键盘，鼠标和屏幕； 传输设备，如网卡和调制解调器。不论这些不同设备的数目巨大，我们只需要理解一些基本的概念，即设备如何加载以及软件如何控制硬件。 基本概念 设备由两部分组成，一个是被称设备为控制器的电器部分，另一个是机械部分。控制器通过系统总线加载到 电脑上。典型的方式是，一组互不冲突的寄存器组被赋予到各个控

制器。I/O端口包含4组寄存器，即状态寄 存器，控制寄存器，数据输入寄存器，数据输出寄存器。状态寄存器拥有可以被主机读取的（状态）位，用来 指示当前命令是否执行完毕，或者字节是否可以被读出或写入，以及任何错误提示。控制寄存器则被主机写操作以启动一条命令或者改变设备的（工作）模式。数据输入寄存器用于获取输入而数据输出寄存器则向主机发送结果。 所以，处理器和设备之间的基本界面是控制和状态寄存器。当处理器执行程序并且遇到与设备相关的指令 时，它通过向相应的设备发送一条命令来执行该指令。控制器执行所要求的动作并设置状态寄存器的特定位，然后进入等待。处理器有责任检查设备的状态直到发现操作完成。例如并口驱动程序（打印机使用的）一般会 轮询打印机以知道打印机是否准备好。如果打印机没有准备好，驱动程序会睡眠一段时间（处理器此时会做其他有用的工作），该过程将重复直到打印机准备好。这种轮询的机制能够改进系统的性能。另外一种方式则是 系统进行不必要的"死等"（unnecessarily waiting）而不做任何有用的工作。 寄存器拥有在I/O空间明确定义的地址范围。通常这些地址在启动时被分配，使用一组在配置文件中定义的参数。各个设备的地址范围可能被预分配，如果设备是静态加载的。这意味内核包含了已存在设备的驱动 程序，以分配的I/O端口能被存放在Proc目录下。你可以在系统使用这些设备时，通过运行“ cat /proc/ioports ”命令同步的检查其所使用的地址范围。第一列输出显示了端口的范围而第二列则是拥用这些端口的设备。一 些操作系统具备在运行时动态加载设备驱动模块的特性。所以任何新的设备都能通过动态加载模块在系统运行时加载

到系统中，并且能够被控制和访问。 设备驱动的概念是非常抽象的并且处于一台计算上所运行软件的最低层。由于直接到设备的硬件特性的限 制。每个设备驱动都只管理一种单一类型的设备。这些类型可能是字符型，快设备型或网络型。如果一个应用程序向设备提出（操作）要求。内核会联系到对应的设备驱动，设备驱动接着向特定的设备发出命令。设备驱 动是一个函数集合：包含了许多调用入口，类似于open，close，read，write，ioctl，llseek 等。当你插入你的模块时，init_module （） 函数会被调用，而模块被移出时，cleanup_module （） 函数会被调用。设备是在 设备驱动的init_module （） 例程中被登记的。 当设备在 init_module （） 中登记时，设备的资源如I/O端口，内存和中断号也在这个函数被分配，这也 是驱动程序能够正确操作设备的需要。如果你分配了任何错误的内存地址，系统会显示错误信息segmentation fault. 而对于I/O端口，系统不会给出任何类似wrong I/O port的信息，但是指派任何现有设备已使用的端口将会造成系统崩溃。当你移出模块时，设备应当被注销，更确切的说，主（设备）号和资源将在cleanup_module （） 函 数中被释放。 设备驱动最频繁的工作时读写IO端口。所以你的驱动应当是确信完美的，被设备使用的端口地址是独占的。任何其他设备都不会使用这段地址范围。为了确认这点，首先驱动应当查明这段地址是否在使用，当驱动发现 这段地址未被使用时，可以申请内核为设备分配这段地址。安全端口分配 现在我们来看看如何通过系统函数来完成资源分配和资源释放。下面的实例 是在linux 2。4内核上进行实验的，以下的所有实现仅适用于Linux操作系统和某些扩展的Unix变

种。 首先侦测可用的端口(地址)范围，通过下面的函数: int check_region (unsigned long start， unsigned long len).函数返回0表示端口地址可用，返回小于零或负的错误编码(-EBUSY or -EINVAL) 表示已在使用中。函数接受2个参数: start 是 连续区域（或I/O端口范围）的起始值，而len是区域内的端口数目。当端口可用时，应该将它分配给设备，通过request_region 函数。 struct resource *request_region (unsigned long start, unsigned long len， char *name). 头两个参数和我们前面看到的一样，字符指针变量name是要分配端口地址的设备名称。函数返回指向resource结构的指针。Resource结构用来描述资源的范围，定义于。结构的格式定义如下: struct resource { const char *name. unsigned long start， end. unsigned long flags. struct resource *parent， *sibiling， *child. }.当模块从内核移出时，端口应当被释放以便为其它设备使用，为此我们在 cleanup_module ( )中 使用release_region ( ) 函数。 函数的语法如下: void release_region (unsigned long start， unsigned long len).两个参数的解释和前面一致。 以上的3个函数实际上是宏定义，定义于。 100Test 下载频道开通，各类考试题目直接下载。详细请访问 www.100test.com