《openEuler内核编程》

课程讲稿

第八章 第1讲

设备管理基础

软件所制

第八章 第1讲 设备管理基础

**学时：1**学时

**教学目的：**系统学习设备驱动基本概念，深入了解设备驱动原理。

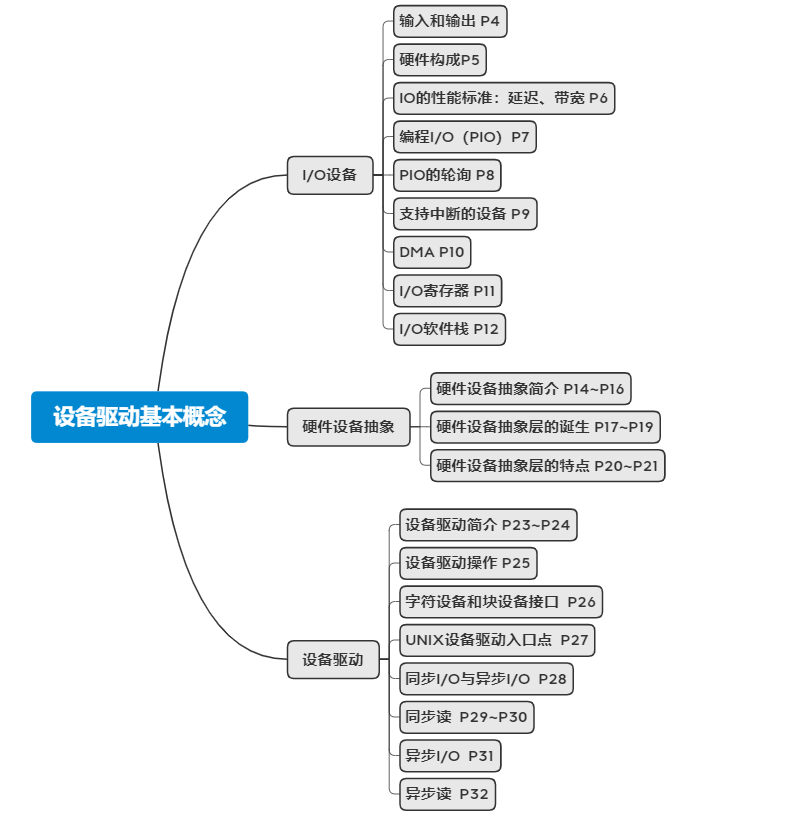
**课程时间线：**



**课外参考读物：**

《LINUX设备驱动程序（第三版）》

**知识框图：**



**PPT讲稿：**

1. 在第八章中我们将介绍关于操作系统，设备管理的相关知识。

2. 这是我们这一章的基本内容，分为五节。在本节课中我们将介绍第一节，内容设备驱动基本概念。

3. 本节之中我们将分为三块内容来介绍设备驱动的基本概念，分别是IO设备、硬件设备抽象和设备驱动，接下来我们先看IO设备的内容

4. 我们先来回顾一下计算机的相关知识。计算机的核心工作是什么？（并不是打游戏），是处理数据。无论你用计算机在做什么，包括浏览网页，使用应用，玩游戏等等一系列的操作，本质都是在处理数据。

那么我们现在来看一下为了让计算机能够正确的执行处理数据的工作，需要哪些部分组成：

首先，核心工作是处理数据，那么我们先来看一下计算相关的硬件有哪些：CPU是核心计算、Cache是缓冲区、内存是快速访问以供CPU调用数据。这三者之间的联系大家应该清楚：本质是内存向Cpu传送数据，但是由于速度问题中间加了一层Cache来作为缓冲。

然后我们既然有了核心工作的组件，那么我们现在就要考虑一个问题了，虽然说我们是从内存获取数据，进入CPU进行数据的处理，最后再返回计算数据给内存，但是内存中的初始数据是哪来的，以及计算机计算完成后给了内存的数据去哪了？

那么就是IO设备的问题了对吧，我们需要Input输入到内存中，然后计算完成后内存传输数据给Output。

那么我们来考虑一下关于IO设备的一系列问题：

IO设备有很多种，存储设备比如硬盘、网络设备比如网线的连接也算是一种IO、显示设备比如显示屏、输入设备比如键盘鼠标、等等等等。我们可以看到这里有很多种不同类型的IO设备，他们之间的区别可能比人和狗的区别还要大（比如说鼠标键盘和网线的区别，如果不是我告诉你他们之间共同点是都属于计算机的IO，那么你可能完全想不到他们之间有什么共同点）。这就是其中一个挑战。

然后我们知道设备其实是需要驱动的，比如你拿着你的鼠标，然后插进了电脑，如果你有留意的话你会注意到其实任务栏的地方有提示：正在安装驱动。那么我们这么多的IO设备几乎每一个就是对应了一个驱动。这也是其中的一个挑战：大量的，繁杂的驱动。

最后一个挑战就是与上面所说的驱动有关了：我们有大量的驱动，然而其实设备驱动往往都是运行在内核态的，如果某个设备驱动出现了BUG，那么往往会引发宕机。这里可能会有同学觉得宕机就是由驱动引发的，和我操作系统有什么关系？那么我问你一个实际情况：你在使用一台电脑，然后你插入了一个USB设备比如说麦克风，然后你的电脑死机了，你会觉得是麦克风垃圾还是电脑垃圾？（顿）那当然都垃圾。所以说作为一个操作系统的层面角度，你是务必要保证自己的正常运行，无论外部的输入是正确还是故意的干扰，你都需要保证自己的正确运行。

那么我们就有点这个感觉了对吧，作为一个操作系统的层面，在IO这一块你需要做到的事情是什么？就是提供一种通用的、一致的、方便的、可靠的方法来访问I/O设备，并且充分发挥IO设备的性能。其中“通用”是对应着第一个挑战：不同类型的设备；“一致”和“方便”对应着第二点，需要大量的设备驱动。然后最后“可靠”对应着第三点：可能出现的问题。

5 然后我们来看一下整体而言的硬件的构成。主要分为计算硬件和IO硬件，其中计算硬件包括CPU 核和 Cache，内存控制器、 I/O 总线逻辑和内存。IO硬件包括I/O 总线或互连、IO 控制器或适配器、I/O 设备。另外还有一些独特的机制包括PIO、中断和DMA，这些虽然是属于一些独特的机制，但是这些机制本身是需要硬件来进行实现的，因此我们在此也将其归入了硬件的构成之中。

那么我们这里可以看到，硬件的构成分为三个大部分的划分：用于计算的硬件、用于IO的硬件以及用于实现特殊机制（功能）的硬件。

6. 我们现在来研究一下IO的性能标准：延迟和带宽，这两个最关键的属性。

开销：就是CPU用于启动设备进行操作的时间，延迟即是传输1字节的时间，实际上延迟=开销+将1字节传输到目的地的时间。

带宽就是启动设备后I/O传输的速率，单位是Bytes/sec。

那么我们看一下右图，这里是不同的设备的带宽情况。我们之前提到需要对IO进行一个一般化的处理，那么我们所做的手段就是如下三点：明确不同的设备有不同的传输速率，这一点上要分开对待。然后不同的设备我们都可以进行对字节传输的抽象，在这一点上每一种设备都是相同的。第三点是我们对不同的IO设备，都以块为传输粒度，这样我们就可以分摊开销了。

7. PIO：我们拿磁盘来说，很早之前，磁盘和内存之间的数据传输是需要CPU控制的，也就是说，如果我们读取磁盘文件到内存中，数据时需要经过CPU存储转发，这种方式称为PIO。我们这里来看一个简单的例子，在RS232串口之中，一个简单的串行控制器包括一个状态寄存器和一些数据寄存器。那么IO的传送过程就是由CPU控制的：等待设备状态变为非“忙” 、 写数据到数据寄存器、通知设备“就绪”设备、等待“就绪”命令、清除“就绪”标志，设置“忙”标志、从数据寄存器中拿走数据，最后清除“忙”标志。

8. PIO的轮询：其实就是在一直等待设备的状态变成非忙，然后才可以进行IO操作。

这样进行处理的好处是简单，逻辑清晰。坏处是很慢，而且对于CPU而言是相当浪费资源的一件事情。我们来举一个例子：如果一个设备的速度是100 ops/sec，CPU需要等待10毫秒，而对于1GHz的CPU，意味着 1千万个CPU 时钟周期。

最后我们需要注意一下，中断机制是可以避免CPU轮询的，这个我们在后面讲，这里提一句。

9. 我们来看一下 如果我们的设备是支持中断的设备会是什么样子：举一个鼠标的例子：·这里是一个简单的鼠标控制器：有一个状态寄存器（完成、中断、……）有一个数据寄存器（ΔX，ΔY按键）。

那么我们进行一次鼠标操作，整个的流程就是这样的：鼠标进行操作后，鼠标控制器等待直到设备状态变为“完成”，将ΔX，ΔY和按键的值保存到数据寄存器、请求中断。那么CPU会响应这个中断处理：清除“完成”标志、将ΔXΔY和按键的值读到内核（变量）、置“完成”标志 最后调用调度器。

我们可以看到在支持中断的设备中这个操作合理了很多，让出来了CPU的时间，当有需要的时候进行中断操作即可。

10. DMA。DMA：后来DMA（Direct Memory Access，直接内存访问）取代了PIO，它可以不经过CPU而直接进行磁盘和内存（内核空间）的数据交换。在DMA模式下，CPU只需要向DMA控制下达指令，让DMA控制器去处理数据的传输即可，DMA控制器通过系统总线来传输数据，传送完毕再通知CPU，这样在很大的程度上降低了CPU占有率，大大的节省了系统资源。而它的数据传输速度其实与PIO相差并不十分明显，因为这主要取决于慢速设备的速度。

整个的一个过程如图所示，我们可以看到在DMA的整个传递过程中，DATA的传递是直接通过总线传输，不需要占用CPU的时间，CPU仅需要进行一次处理即可不用再对这个数据处理进行更多的操作，对CPU而言可以说是极大的降低了IO对CPU的负载。

11. IO寄存器，作用是将设备寄存器映射到物理内存。

好处是可以简单、统一的对设备进行处理，设备寄存器和内存使用相同的CPU指令来访问。也就是说可以把设备寄存器作为一个并不特殊的存在，这样对于进行统一的处理过程中就没有必要进行特殊化了。

问题是虽然看起来不特殊，但是还是有一点点特殊的：这些“内存区域”是不能被缓存的。解决的办法就是把他们设置为“not cachable”就可以了。

12．这里是软件栈的一个情况，从栈顶至低分别是用户级lO、软件与设备无关的OS软件、设备驱动程序、中断处理程序、硬件  
13. 然后我们来看本节内容的第二部分：硬件设备抽象。

14. 我们来看一下这里是简单的一套计算机，我们知道操作系统其实是虚拟运行在主机之中的特殊的计算机程序，那么我们思考一下这个问题：作为一个虚拟运行的程序，它能够与其他程序进行交互我们是能够理解的，但是你把眼光放大一点，看图之中包括显示屏、键盘、鼠标以及图中没有画出来的麦克风、耳机，操作系统作为一个虚拟的东西，它是如何与这些硬件设备交互的呢？

15. 那么其实扮演了让虚拟的软件与实际存在的硬件进行交互的就是硬件设备抽象。硬件抽象层是位于操作系统内核与硬件电路之间的接口层，其目的在于将硬件抽象化。它隐藏了特定平台的硬件接口细节,为操作系统提供虚拟硬件平台,使其具有硬件无关性,可在多种平台上进行移植。 从软硬件测试的角度来看，软硬件的测试工作都可分别基于硬件抽象层来完成，使得软硬件测试工作的并行进行成为可能。

16. 我们可以看到，在应用程序端进行调用Api的时候进行调用一系列正常函数（无论在什么样的设备上都可以使用这些函数），然后通过HAL的API映射，在硬件设备的调用时就会转化成真正的调用函数。

17. 那么我们现在来看一下HAL诞生的原因：硬件抽象层（HAL）是对linux驱动的一个封装:对上层提供统一接口，上层应用不必知道下层硬件具体怎么实现工作的，它屏蔽了底层的实现细节。

18. 传统的Linux：对硬件的所有操作均在内核空间中的linux驱动中实现。而在引入HAL后：对硬件的操作分为HAL和linux驱动两部分，其实相当于是把HAL从linux驱动中剥离出来，自成一体。那么剥离出来后有一个重要的区别就是HAL是属于用户空间，linux驱动属于内核空间。

19. 为什么要剥离出来？我们可以看到有两方面原因，第一方面是我们之前说到过的API对API之间函数的映射，在谷歌统一规范下的HAL的框架，为上层framework通过JNI调用HAL提供了统一的API，硬件开发商或者移植人员只需要按照框架开发即可，无需话费精力在与上层的交互上的实现上，将精力放在HAL层本身的实现上即可。也就是说HAL两边，一边是调用硬件的软件，一边是接受调用的硬件，两方面都可以按照现有的框架来进行编程，无需互相沟通，这样不仅是省下来了沟通的时间，同时也对可拆卸性得到了极大的满足。

另一方面就是我们刚刚所提到的，剥离出来后有一个重要的区别就是HAL是属于用户空间，linux驱动属于内核空间。相当于u我们将HAL本来是在内核态，现在我们将其移入了用户态。从这个角度来看我们是解放了内核态的一部分内容，那么将其从内核态解放出来有什么意义呢？我们知道，从商业角度，许多硬件厂商不愿意将自己硬件相关一些核心的东西开源出去，假如将对自己硬件的驱动程序全部放入内核空间驱动程序实现，那么必须遵循GPL协议，是必须开源的。有了HAL层之后，他们可以把一些核心的算法之类的东西的实现放在HAL层，而HAL层位于用户空间，不属于linux内核，和android源码一样遵循的是Apache协议，这个是可以不用开源的。

20. 我们对HAL进行一下内部的划分可以分成这几个部分：上层软件、虚拟驱动：设置管理模块、内部通信SERVER、内部以太网、内部通信CLIENT、用户接入口。

21. 我们来介绍一下相关的特点：硬件抽象层具有与硬件的密切相关性。硬件抽象层具有与操作系统无关性。接口定义的功能应包含硬件或系统所需硬件支持的所有功能。接口定义简单明了，太多接口函数会增加软件模拟的复杂性。具有可测性的接口设计有利于系统的软硬件测试和集成。

22. 我们来看本节课的第三部分：设备驱动。

23. 这张图是我们操作系统与硬件进行交互的一个展示。可以看到中间这个橙色的部分，就是我们设备驱动，承担了二者之间进行交互的枢纽。

24. 那么作为设备驱动，主要的任务就是给操作系统的其它模块提供API：比如说Init初始化，Open启动，Close关闭，read write读写操作。

除此之外，它还需要与设备控制器打交道：与设备控制器交互以进行传输包括命令、参数、数据在内的数据

作为驱动程序，它的操作分为如下几种：初始化设备、解析OS传来的命令、多个请求的调度、管理数据传输、接收和处理中断、维护驱动与内核数据结构的完整性。

25. 这里是我们之前提到的API：Init：初始化硬件；Open：初始化驱动并分配资源；Close：清除，回收资源，并可关闭设备。

而read和write对于不同的设备操作是不同的。因此我们这里引出了设备驱动的不同类型：设备驱动分为四种不同的设备：字符设备，可以进行可变长度的数据传输； 块设备，以固定大小的块为粒度的数据传输；终端设备，字符驱动+终端控制；网络设备，也就是网络数据流。之后我们会进行对不同设备的详解，在这里我们仅对不同的设备类型就这些操作进行不同的解析：

26. 对于字符设备接口而言，read和write分别是从字节流设备上读“size”字节数据和将“bufferAddr”中size字节数据写入字节流设备。

对于块设备接口而言，其是以固定大小的块为粒度的数据传输，因此read和write均以块为单位即可，无需指定大小。

27. 我们来关注一下在Unix设备驱动中的一个流程：

init()：初始化硬件；然后start()，开机时初始化，需要系统服务。然后open（dev，flag，id）和 close（dev，flag，id），进行初始化资源和释放资源；然后是halt()，在系统关机前要调用此函数。到这里是一个正常的开机与关机流程。

然后下面的是一些特殊的函数：intr（vector)在发生硬件中断时由内核调用；read（…）和 write（…)就是之前所说的数据传输。select（pri）函数在内核中每秒调用25次~100次，其的作用就是把当前的文件指针挂到等待队列。最后是一个特殊函数ioctl（dev，cmd，arg，mode）是进行特殊请求处理的函数。

28. 接下来我们来看同步和异步IO的情况。

在同步IO之中，read( ) 和 write( )将阻塞用户进程，直到读写完成。在一个进程做同步I/O时， OS调度另一个进程执行。

而在异步I/O之中，read( ) 和 write( )不阻塞用户进程，在I/O完成以前，用户进程可以做别的事儿，当I/O完成后将通知用户进程。

29. 我们可以看到这里是一个同步读的过程，整个流程比较简单，但是代价是用户进程被阻塞，只能执行另一个进程。

30. 在同步读的过程中，是这样的一个流程：进程调用read( )系统调用，然后系统调用代码检查正确性和缓存，然后如果需要进行I/O，调用设备驱动程序，然后设备驱动程序为读数据分配一个buffer，并调度I/O请求，然后启动DMA做读传输，然后阻塞当前进程，调度一个就绪的进程，然后设备控制器进行DMA读传输，然后传输完时，设备发送一个中断请求，然后中断处理程序唤醒被阻塞的进程（将它加入就绪队列），然后将数据从内核buffer拷贝到用户buffer，然后系统调用返回到用户程序，最后用户进程继续执行。

31.异步IO。在异步IO之中，有如下一些函数：aio\_cancel: 取消异步读写请求aio\_error: 获取异步I/O错误状态；aio\_fsync: 异步地将缓存脏块写回磁盘，并将errno设置为ENOSYS；aio\_read: 异步读；aio\_return: 获取异步I/O操作的状态；aio\_suspend: 挂起直到异步I/O操作完成；aio\_write: 异步写；lio\_listio: 提交一组I/O请求

32.在异步读的流程中我们可以看到，流程之中复杂了不少，但是流程复杂带来的效果是非常大的提升，可以解放用户进程，在IO的过程中做别的事情。