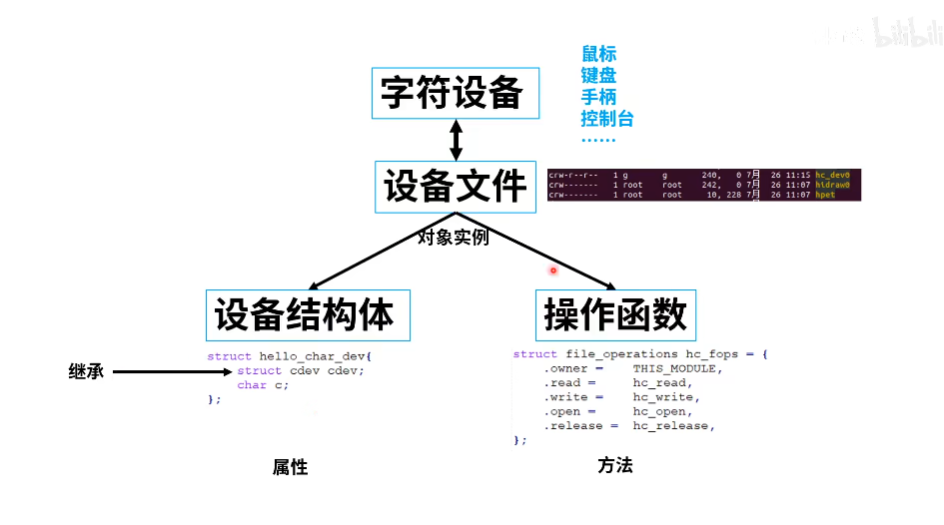
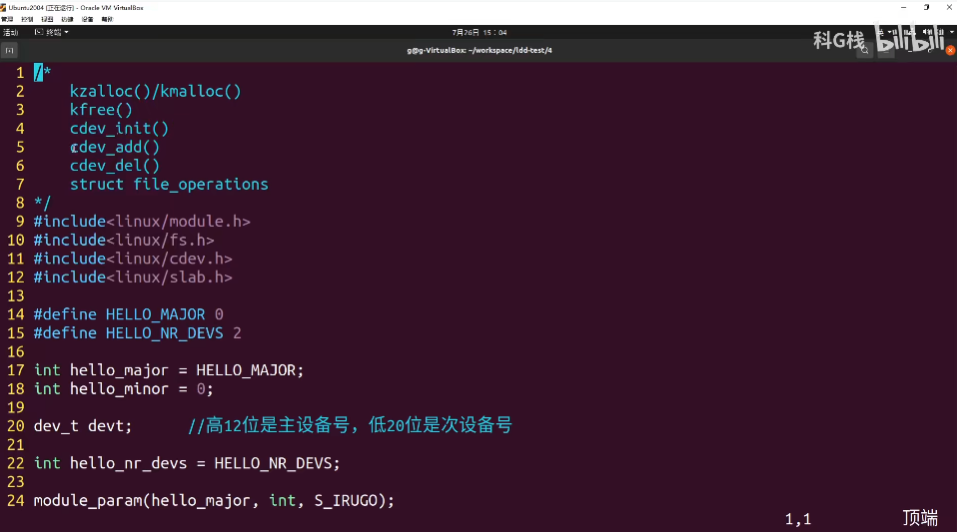
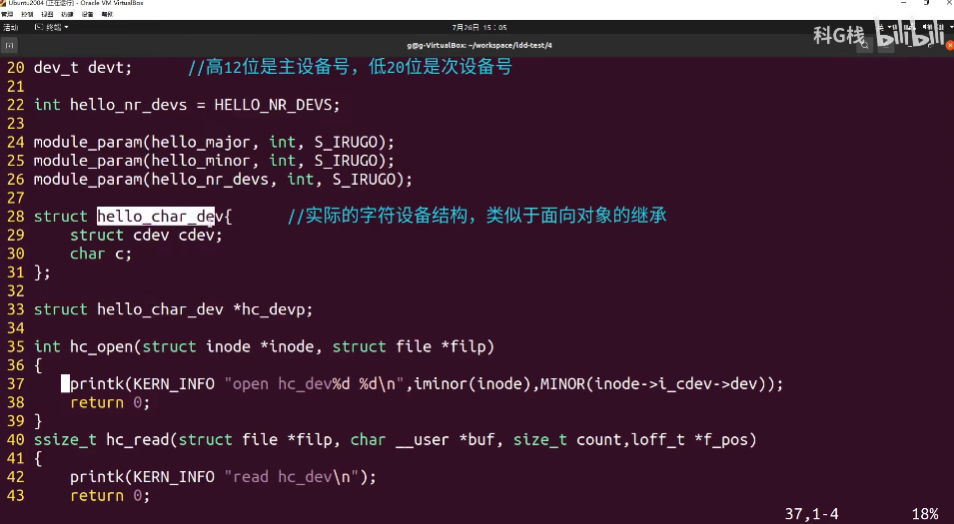
参考

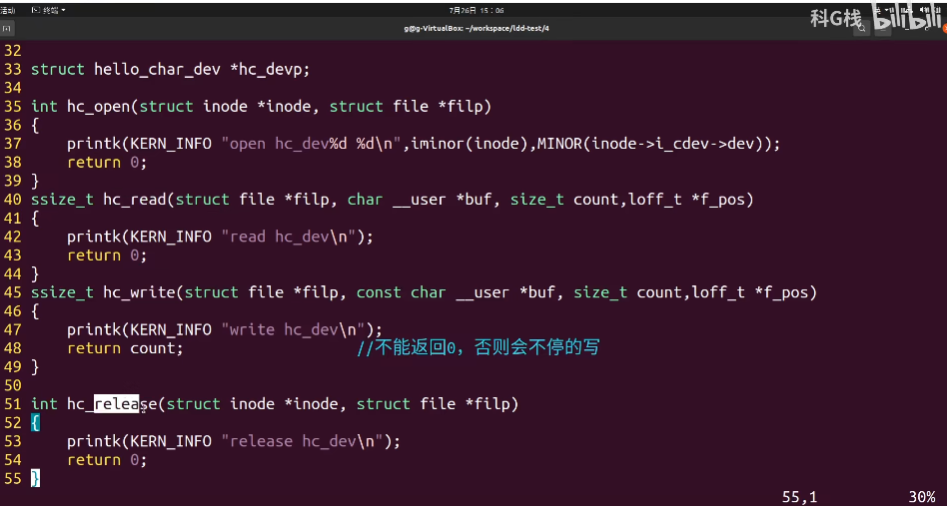
https://www.bilibili.com/video/BV1n54y1J7vM/?spm\_id\_from=333.337.search-card.all.click&vd\_source=4a018e55005b433dc9d72a8e969a1c5f

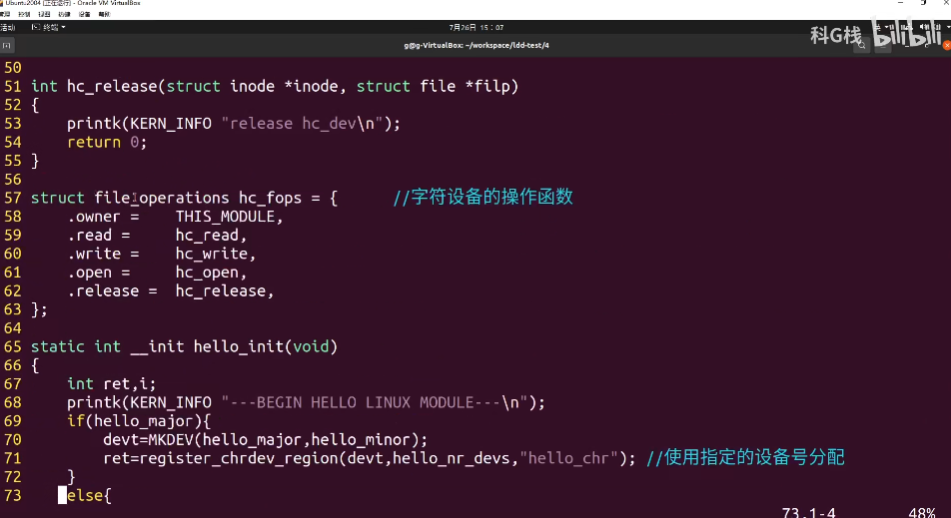
https://github.com/gyxkgz/ldd-test











嗨大家好

这期视频给大家介绍一下

如何写一个最简单的字符设备驱动

或者可以理解为字符设备驱动的一个框架

它不实现任何具体的业务逻辑

这样我们可以更好地理解

如何编写一个字符设备驱动

在看具体的代码之前

我们先来看一张图

在LINUX当中使用到了很多面向对象的概念

设备驱动就是一个非常典型的例子

我们在控制外部设备的时候

一般都是通过读写device下的设备文件来实现的

从这个角度来看

设备文件就相当于是外部实际设备的一个抽象

对于一个通常的设备

我们会用一个设备结构体

来描述这个设备的属性

利用一个叫file operations的结构来管理

操作这个设备的函数

从面向对象的角度来看

这个设备结构体就是这个设备的属性

操作函数就是这个设备的方法

在这个设备结构体当中

有一个struct c device的结构

这个结构是LINUX抽象出的

所有字符设备的共有属性

在我们的字符设备结构体中

一定要包含这个结构体

这样我们就有了所有字符设备的共有属性

从面向对象的角度看

这有点类似于继承关系

所以从面向对象的角度来理解

设备驱动对于我们编写代码来说会非常有帮助

下面我们来看一下具体的代码

在这个代码中

首先我们会涉及到几个新的函数

ky look和key lock是用来分配内核存储空间的

后面我们需要用到这些函数

来给我们的设备结构来分配存储空间

KFREE是释放分配的内核空间

c device init是初始化字符设备

CD vs aid是将字符设备添加到内核当中

CD vs d e l移除添加到内核当中的字符设备

Structed file operations

这个结构就是用来保存操作字符设备的函数的

对于头文件

我们需要额外增加c device的H和slept h

头文件

CD vs dh就是字符设备的头文件

slap点H是分配存储空间的头文件

再往下面的这些变量

在我之前的视频中已经介绍过了

这里我就不介绍了

这个STRUCT

Hello

Child device

就是我们新建的字符设备

它里面包括了这个struct c device

以及我们自己定义的一些属性或者变量

这里为了简便

只给它添加了一个字符变量

没有什么实际的作用

只是为了演示

在下面

我们定义一个指向这样一个结构类型的指针

后面我们会不会给这个指针来分配存储空间

这样一个字符设备的属性就构建完成了

接下来我们为这个字符设备构建操作函数

实际的字符设备可能有很多个操作函数

这里我们简化只提供了四个操作函数

分别为open read

write和release

顾名思义

open就是打开这个设备

read就是读取设备

write是写设备

release是释放设备

在这些方法当中

我们也不实现任何具体的业务逻辑

只是去打印一些信息

我们只要知道调到这些函数就OK了

在这个right函数这里要注意返回值不能是零

如果返回值为零

则会不停的去调用这个right函数

然后把这些函数填充到这个file operation结构体当中

就OK了

这样我们字符设备的属性以及方法

就都构建完成了

在这个模块的初始化函数里

前面这部分就是用来申请字符设备的设备号的

之前我已经讲过了

这里就不赘述了

下面使用K在LCK这个函数

来为这个字符设备结构体分配存储空间

空间的大小是这个字符设备结构的大小乘以

需要分配的设备个数

这里这个变量值是二

所以我们分配了两个这样的字符设备

gfp kernel是一个标志

在内存中分配存储空间

通常使用这个标志

k i logo的意思是分配空间

并且将空间置为零

下面我们来判断一下是否分配成功

如果不成功

做相应的失败处理

再下面就是具体的对字符设备的操作

这里我们要分配两个字符设备

所以使用了一个for循环

首先我们要初始化字符设备

用c device init这个函数

它的第一个参数是STRUCT

c device结构的一个指针

这里具体来说就是我们定义的字符设备中

c device成员的地址

它的第二个参数就是操作这个字符设备的方法

那这里就是这个file operations结构的地址

下面这一行是因为在c device中有一个honor成员

我们将这个成员给它复制成this model这个红

再下面就是将这个字符设备

添加到内核当中

使用c device a

它的第一个参数依然是指向CD vs结构的指针

在这里依然是我们结构下CDBS成员的地址

第二个参数就是设备的设备号

这里我们使用主设备号

次设备号

加索引号构建出这个设备号

最后一个是计数

这个通常是1cd vs a的函数

执行以后

我们的这个字符设备就真正添加到了内核当中

就可以对它进行操作了

下面这里判断是否添加成功

不成功的话

我们提示一个信息

这里我顺便提一下LINUX当中的错误处理

在我们的C语言编程中

一般不推荐使用go to

但是在内核当中

对于错误的处理

使用go to反而会更加方便和灵活

比如说这里如果分配设备号失败

我们可以使用go to到file处执行return返回

那如果你分配存储失败

我们执行go to failer cheeoc

到这里我们执行一些错误处理

错误处理的执行是有顺序的

他的原则是发生错误的时候

要将发生错误之前获得的资源释放掉

举例来说

就是如果在分配存储空间的时候发生错误

那至少说明我们申请的字符设备号是成功的

所以一旦发生分配存储空间失败

我们首先要释放掉之前获得的设备号

然后再执行这个return返回

这种错误处理的方式

既方便又高效

在内核中常常会见到

下面是模块的清除函数

在清除函数当中

首先我们要将添加到内核当中的字符设备

一一将它移除

使用CD vs t a l

它的参数同样是字符设备的指针

我们这里传入的就是我们字符设备当中

c device成员的地址

然后使用k free

将我们之前分配的内存空间释放掉

然后释放掉我们申请得到的设备号

这样一个最简单的字符设备驱动就构建完成了

下面我们具体编译来看一下

我们将它插入到内核当中

DMC值来看一下

OK这样就是初始化OK了

然后我们看一下它的设备号

A240

Hello

Child

说明是OK的

此时我们的设备驱动程序已经可以工作了

但是我们却发现在device下

没有这样一个操作的接口

这是因为我们默认的这种方式

系统不会在这个device下自动生成

对应的设备文件

需要我们手动去创建下一个视频

我会介绍如何让系统自动为我们创建设备文件

这期视频我们用手动的方式来创建

创建的方法是使用make node这样一个命令

第一个参数是device下我们需要创建的设备名称

然后是一个C符号

然后是主设备号

然后是次设备号

确定

这样我们就添加了一个我们自己的字符设备

之前我们在驱动中创建了两个

所以我们正常再创建一个

这个是一主设备号一样

次设备号不一样

我们在JVS下看一下

嗯这样我们就添加了两个字符设备文件

这两个文件就跟我们在驱动中申请的字符设备

是对应上的

下面我们来对这个设备进行一些简单的操作

我们用cat来去读它

先读一下零吧

没有任何提示

说明我们读取成功了

用DEMESSAGE来看一下

这里它打印出了open read release

说明通过这个cat命令

他首先打开了这个设备

然后读取了这个设备

读完以后又释放了这个设备

也就是关掉了这个设备

在open的时候

我同样给它添加了一个标志

原来标志它现在是读的设备一还是设备二

下面我们来读一下第二个设备

第二个设备是device1

然后我们再用DEMESSAGE来看一下

这里他打开的时候就变成了device1

说明我们成功的读到了第二个设备

下面我们来向设备写来看一下

用一口随便写点东西到第一个设备当中

他这里提示权限不够

这里为了能实现写设备

我们将这个设备改到我们用户下使用

Change honor

这样就将这两个设备改到了我们当前的用户下

继续用ECO

我们来试一下

看能不能写它

没有任何提示

说明我们写成功了

DMC值来看一下

这里他先打开了设备零

然后调用了写函数

然后关掉设备

你再写一下设备一

同样也成功了

这里他打开了设备一

然后写

然后关掉

OK以上就是这期视频的所有内容

我们实现了一个最最最简单的字符设备

后续我们会不断的完善和增强

这个字符设备的功能

好了这期视频就到这里了

希望对小伙伴们有所帮助

如果对你有帮助的话

就点赞支持我一下吧

感谢大家的观看

我们下期再见啦

拜拜