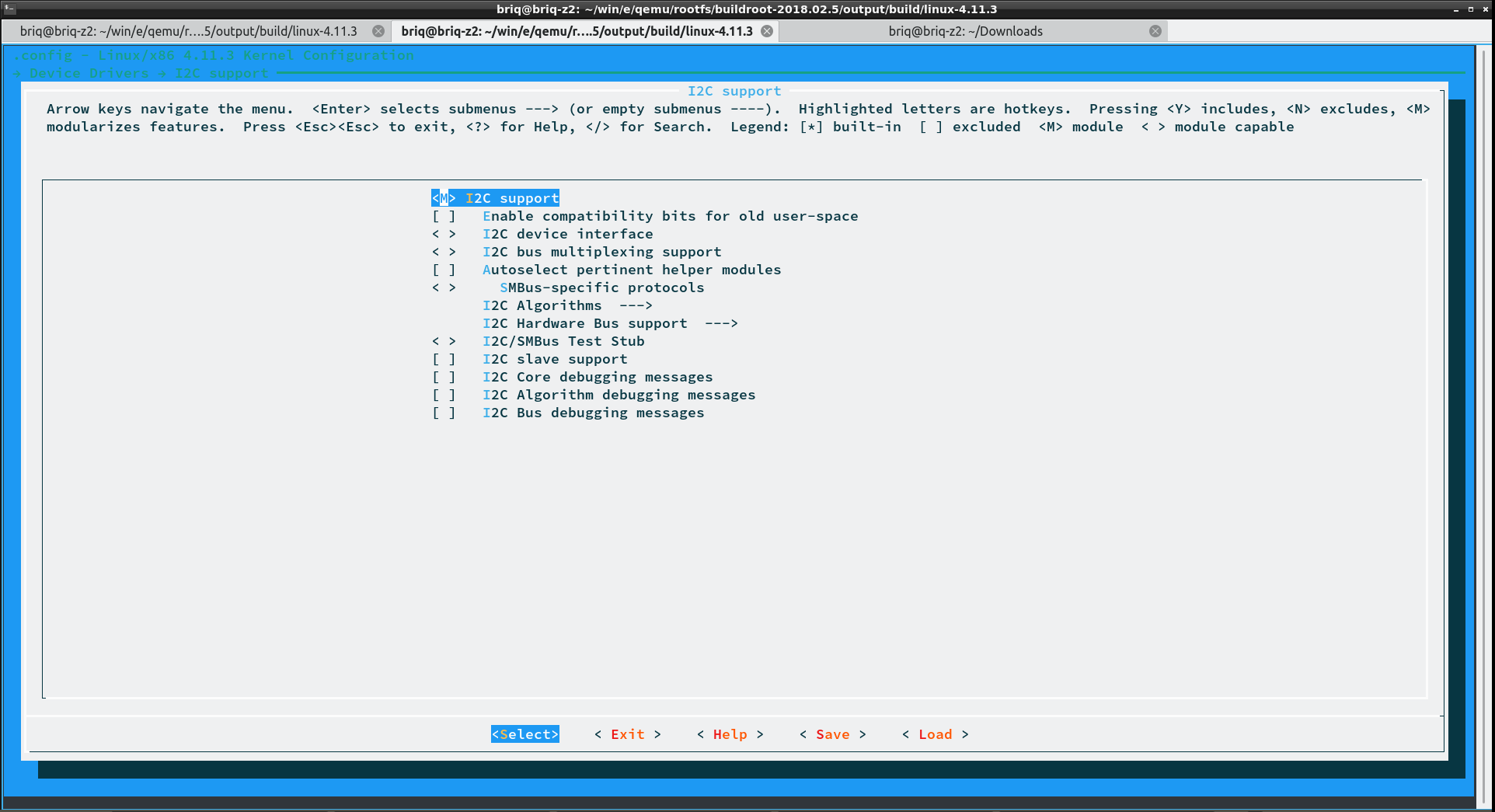
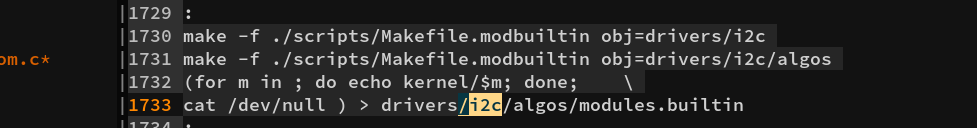
准备学习编写linux driver问题，但是一开始，不知道驱动是怎么编译的，所以就在linux源码树中，研究了下如何编译c文件成模块的

**1、首先以最简单的i2c驱动为例，我使用make menuconfig，修改i2c驱动编译成模块，并保存，详细如下图所示：**



**2、然后使用make -w -n modules > debug\_make\_modules\_i2c.log捕获执行make modules后都干了什么， -w，表示make过程中如果改变了路径去调用其他makefile的话，会打印该路径，以便跟踪调试，-n，表示只是模拟过程，不真正的执行makefile中的实际编译过程。**

打开debug\_make\_modules\_i2c.log文件， 搜索关键字drivers/i2c, 定位到drivers/i2c目录下的i2c驱动编译信息，如下所示：



发现了重要信息，那就是make -f ./scripts/Makefile.modbuiltin obj=drivers/i2c这条语句，原来，在源码树下执行make modules，会调用scripts/makefile.modbuiltin这个makefile，里面包含了怎么去编译drivers/i2c目录下的c文件

**3、我们继续看makefile.modbuiltin文件，看看如何编译模块的**



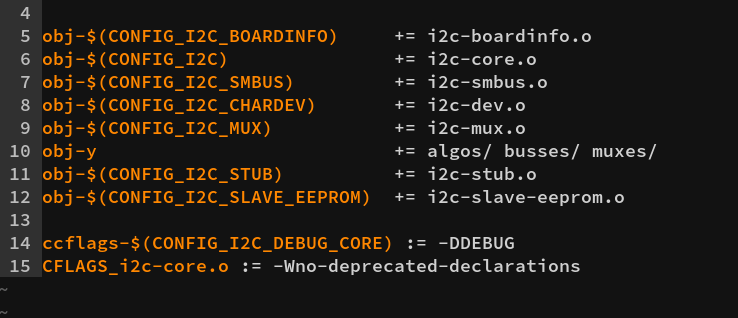
make -f ./scripts/Makefile.modbuiltin obj=drivers/i2c，在上图中的第5行，src就是drivers/i2c目录，再看第24行这条语句，执行过后，kbuild-file就是drivers/i2c/Makefile，

第25行，表示去执行drivers/i2c/Makefile，那么这个Makefile文件里面是什么了？

打开drivers/i2c/Makefile看看都有什么内容。

该文件，全是obj-$(CONFIG\_I2C)的变量赋值，那么CONFIG\_I2C这个变量，我们知道在make menuconfig中我们定义的是M，那么CONFIG\_I2C=M,扩展出来就是：

obj-M = i2c-core.o等一系列的定义



再回到makefile.modbuiltin文件中，在该文件中，就会使用obj-M变量，完成i2c-core.c

文件的编译，编译成模块。

但是我们在外部编译驱动，使用的是makefile.modpost，而不是makefile.modbuiltin脚本。

**4、以下摘抄网络，描述了编译驱动的makefile执行流程，**

# **linux设备驱动makefile入门解析**

2013年03月01日 15:40:31 [18553514996](https://me.csdn.net/shanzhizi" \t "/home/briq/Documents\\x/_blank) 阅读数：14824更多

个人分类： [Linux](https://blog.csdn.net/shanzhizi/article/category/781010" \t "/home/briq/Documents\\x/_blank)

版权声明：本文为博主原创文章，未经博主允许不得转载。 https://blog.csdn.net/shanzhizi/article/details/8626474

以下内容仅作参考，能力有限，如有错误还请纠正。  
对于一个普通的linux设备驱动模块，以下是一个经典的makefile代码，使用下面这个[makefile](http://blog.csdn.net/shanzhizi" \t "/home/briq/Documents\\x/_blank)可以  
完成大部分驱动的编译，使用时只需要修改一下要编译生成的驱动名称即可。只需修改obj-m的值。  
  
  
ifneq ($(KERNELRELEASE),)  
obj-m:=hello.o  
else  
#generate the path  
CURRENT\_PATH:=$(shell pwd)  
#the absolute path  
LINUX\_KERNEL\_PATH:=/lib/modules/$(shell uname -r)/build  
#complie object  
default:  
make -C $(LINUX\_KERNEL\_PATH) M=$(CURRENT\_PATH) modules  
clean:  
make -C $(LINUX\_KERNEL\_PATH) M=$(CURRENT\_PATH) clean  
endif  
  
  
说明：  
当我们在模块的源代码目录下运行make时，make是怎么执行的呢？  
假设模块的源代码目录是/home/study/prog/mod/hello/下。  
先说明以下makefile中一些变量意义：  
（1）KERNELRELEASE在linux内核源代码中的顶层makefile中有定义  
（2）shell pwd会取得当前工作路径  
（3）shell uname -r会取得当前内核的版本号  
（4）LINUX\_KERNEL\_PATH变量便是当前内核的源代码目录。  
关于linux源码的目录有两个，分别为"/lib/modules/$(shell uname -r)/build"和"/usr/src/linux-header-$(shell uname -r)/"，  
但如果编译过内核就会知道，usr目录下那个源代码一般是我们自己下载后解压的，而lib目录下的则是在编译时自动copy过去的，  
两者的文件结构完全一样，因此有时也将内核源码目录设置成/usr/src/linux-header-$(shell uname -r)/。关于内核源码目录  
可以根据自己的存放位置进行修改。  
（5）make -C $(LINUX\_KERNEL\_PATH) M=$(CURRENT\_PATH) modules  
这就是编译模块了：首先改变目录到-C选项指定的位置（即内核源代码目录），其中保存有内核的顶层makefile；  
M=选项让该makefile在构造modules目标之前返回到模块源代码目录；然后，modueles目标指向obj-m变量中设定的模块；  
在上面的例子中，我们将该变量设置成了hello.o。  
  
  
  
按照顺序分析以下make的执行步骤：  
在模块的源代码目录下执行make，此时，宏“KERNELRELEASE”没有定义，因此进入else。  
由于make 后面没有目标，所以make会在Makefile中的第一个不是以.开头的目标作为默认的目标执行。  
于是default成为make的目标。  
make会执行 $(MAKE) -C $(KERNELDIR) M=$(PWD) modules ,假设当前内核版本是2.6.13-study,  
所以$(shell uname -r)的结果是 2.6.13-study ，这里实际运行的是  
make -C /lib/modules/2.6.13-study/build M=/home/study/prog/mod/hello/ modules  
  
  
-C 表示到存放内核的目录执行其makefile，在执行过程中会定义KERNELRELEASE，  
然后M=$(CURDIR)表示返回到当前目录，再次执行[makefile](http://blog.csdn.net/shanzhizi" \t "/home/briq/Documents\\x/_blank)，modules表示编译成模块的意思。  
而此时KERNELRELEASE已定义，则会执行obj-m += hello.o，表示会将hello\_world.o目标编译成.ko模块。  
若有多个源文件，则采用如下方法：  
obj-m := hello.o  
hello-objs := file1.o file2.o file3.o  
关于make modules的更详细的过程可以在内核源码目录下的scripts/Makefile.modpost文件的注释 中找到。  
  
  
如果把hello模块移动到内核源代码中。例如放到/usr/src/linux/driver/中， KERNELRELEASE就有定义了。  
在/usr/src/linux/Makefile中有KERNELRELEASE=$(VERSION).$(PATCHLEVEL).$(SUBLEVEL)$(EXTRAVERSION)$(LOCALVERSION)。

这时候，hello模块也不再是单独用make编译，而是在内核中用make modules进行编译，此时驱动模块便和内核编译在一起。

关于内核模块编译可以参考另外一篇文章：[linux内核模块编译基础知识](http://blog.csdn.net/shanzhizi/article/details/8626419" \t "/home/briq/Documents\\x/_blank)。

转载请注明：[http://blog.csdn.net/shanzhizi](http://blog.csdn.net/shanzhizi" \t "/home/briq/Documents\\x/_blank)