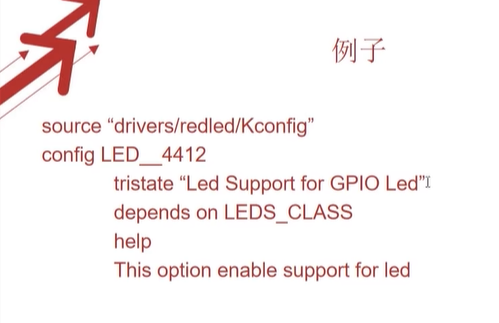


将驱动编译进内核，在内核启动的时候就加载这个模块。

由上一节知：留给我们可以改动的就是KConfig和Makefile文件。

在修改Kconfig之前，我们先分析一个例子：



1.

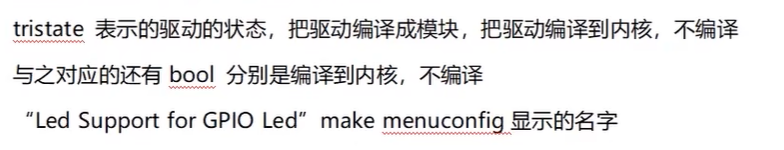
IMG_256

2.

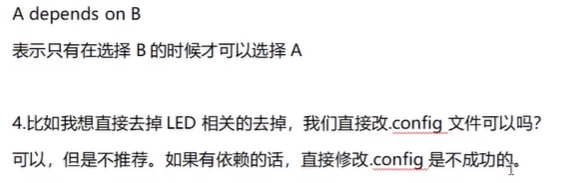
IMG_256

它的全称是CONFIG\_LED\_442只不过他把CONFIG省略了。（了解即可）

3.



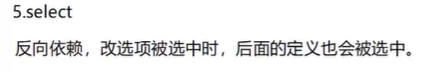
4.



上面这就是为什么不直接修改.config文件： .config是根据Kconfig文件配置而得到的配置文件。比如说想要去掉LED相关的驱动。我们直接去改写.config文件虽然是可以的，但是不推荐。

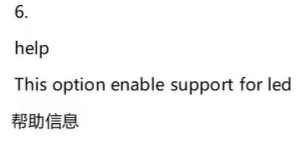
因为有的选项是有依赖的，比如说：上面这个例子就是依赖于 LEDS\_CLASS文件。所以只有当LEDS\_CLASS被选中的时候，这个选项才可以被选中。如果直接修改 .config文件，如果修改的地方有依赖的话，直接修改是不成功的。

5.



反向依赖。该项被选中的时候，select后面定义的那一项也会被选中。

6.

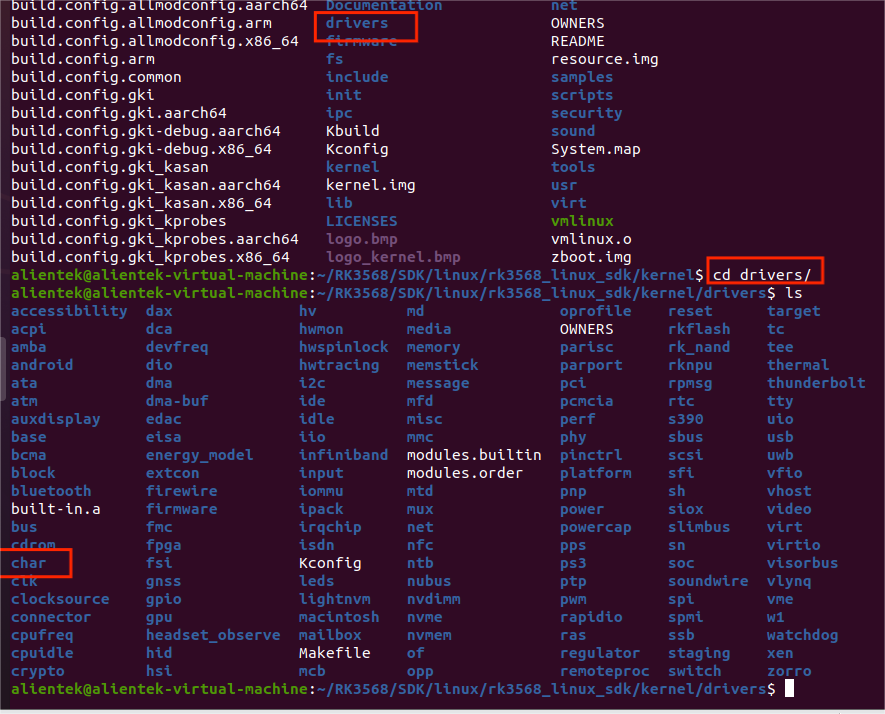


——————————————————————————————————

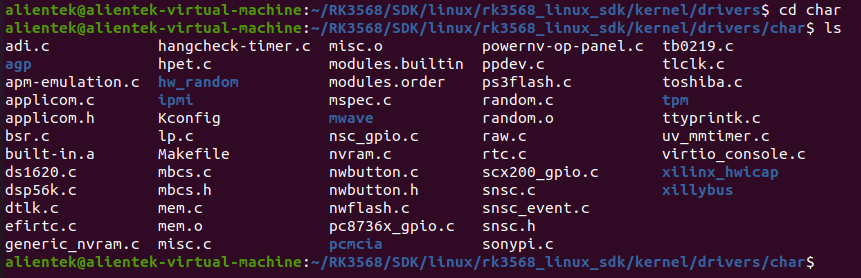
实操：

先进入到内核源码的路径下：这里要把上节课写的驱动放到drivers/char目录下：

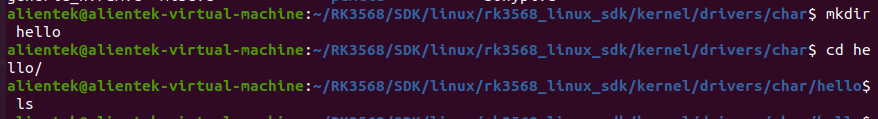
1.进入kernel/drivers/char目录

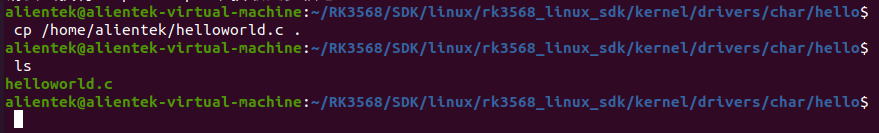


这里：char就是字符设备。

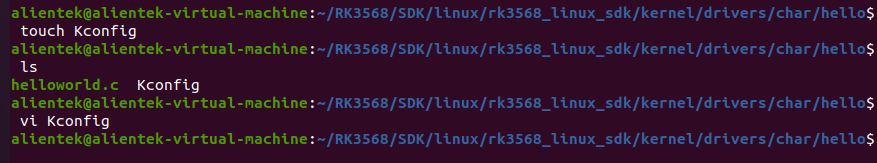


2.创建一个hello文件夹（用于存放helloworld.c文件）。进入该文件夹后将helloworld.c拷贝到该路径下：



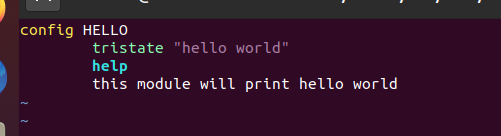


1. 此时需要创建一个Kconfig文件：



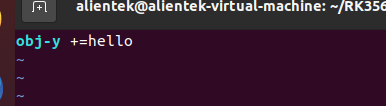
Kconfig的内容仿照上面的例程编写：

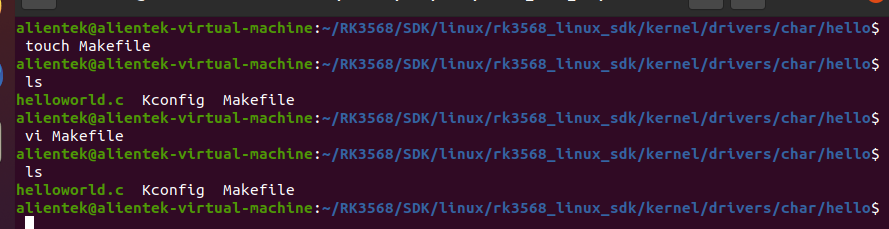
此处不添加依赖。并且给予3种状态。



1. 有了Kconfig之后，还需要创建一个Makefile文件：

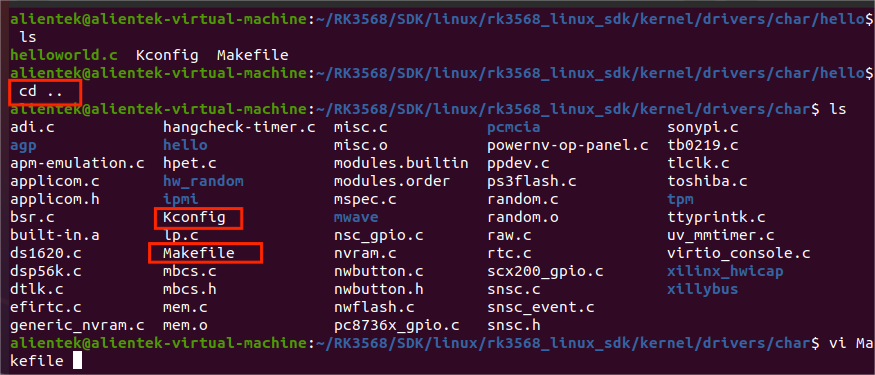
Makefile的内容如下：(他写错了暂时不写内容，应该写在上一级的目录)





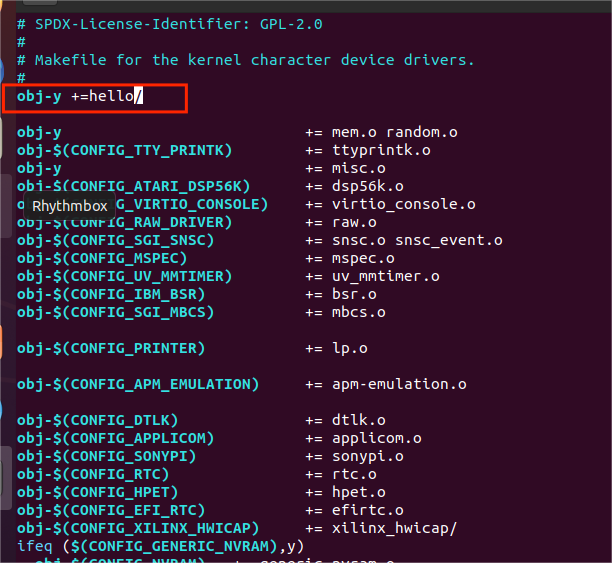
此时文件便准备齐全了。现在就只需要将hello驱动包含进去即可：

要包含进去，那么就需要修改上一级的Makefile和Kconfig。



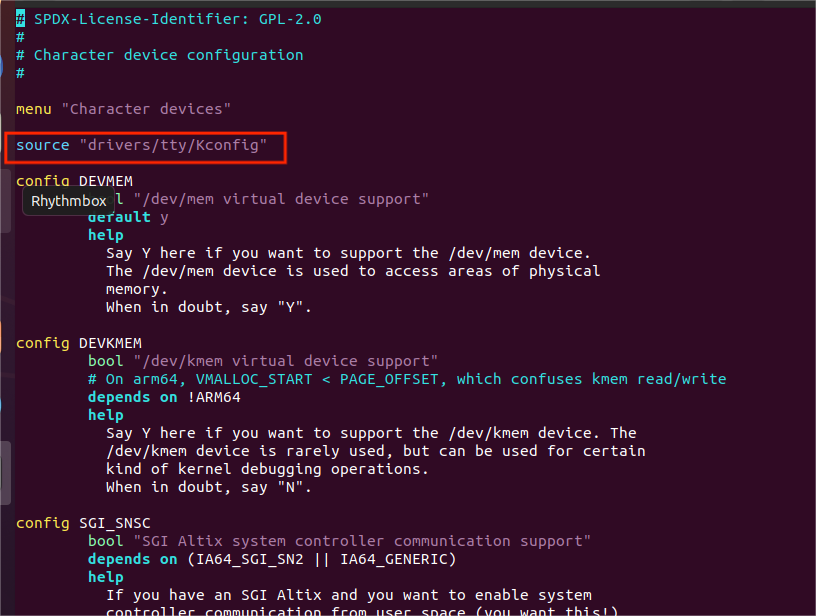
1. （此处，修改了第4点，因为他Makefile写错了，不应当写在hello文件夹下，实际代码应该在hello的同级目录的Makefile里面）





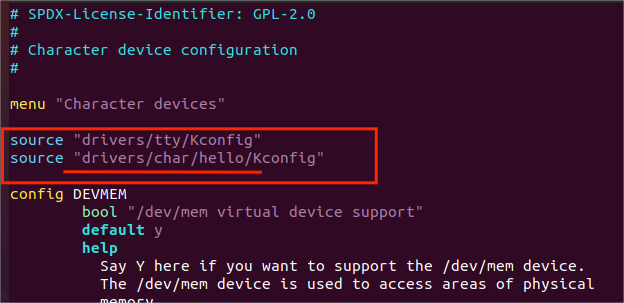
1. 再打开Kconfig文件：

在Kconfig文件里面，我们可以看到如下的一句代码：



他就表示：包含了 drivers/tty下面的东西

我们将这句话复制，然后改写成：drivers/char/hello，然后保存退出即可：



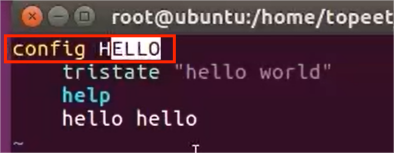
1. 再次进入hello文件夹下，来修改：之前写错的Makefile:

IMG_256

CONFIG\_HELLO是menuconfig时候配置选项的名字，menuconfig会自动把前面的CONFIG省略掉，但我们在编写Makefile的代码的时候是不能省略的。

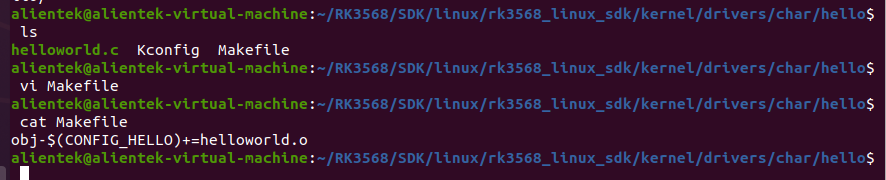
那么这个变量就会根据我们选择的状态来改变，比如:我们在menuconfig里面选择的是编译进内核里面，这个变量(CONFIG\_HELLO)的值 就等于Y。也就是整体就是obj-Y。选择编译成模块的话就是obj-M。

而CONFIG\_HELLO这个名字是怎么来的呢？我们打开Hello目录下的Kconfig：



这里是把CONFIG省略掉的，我们自己加上就成了Makefile里面写的名字:CONFIG\_HELLO了。

Makefile修改后的内容为：

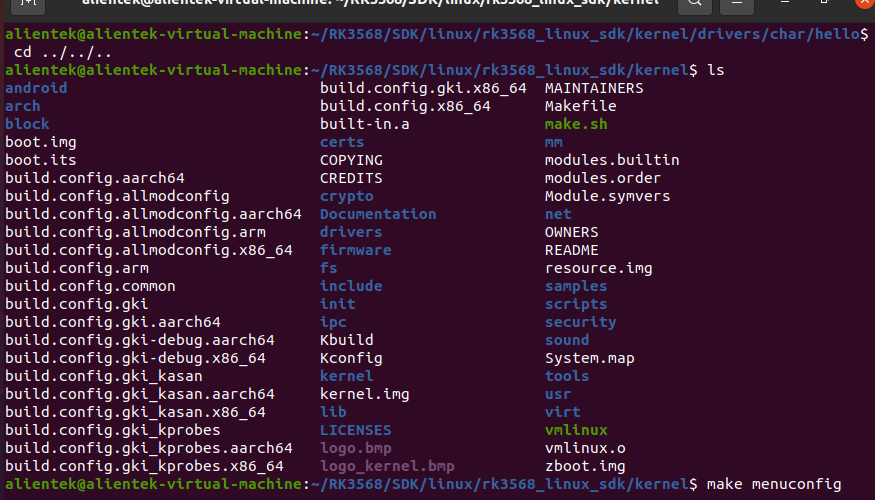


1. 我们返回到内核目录下：

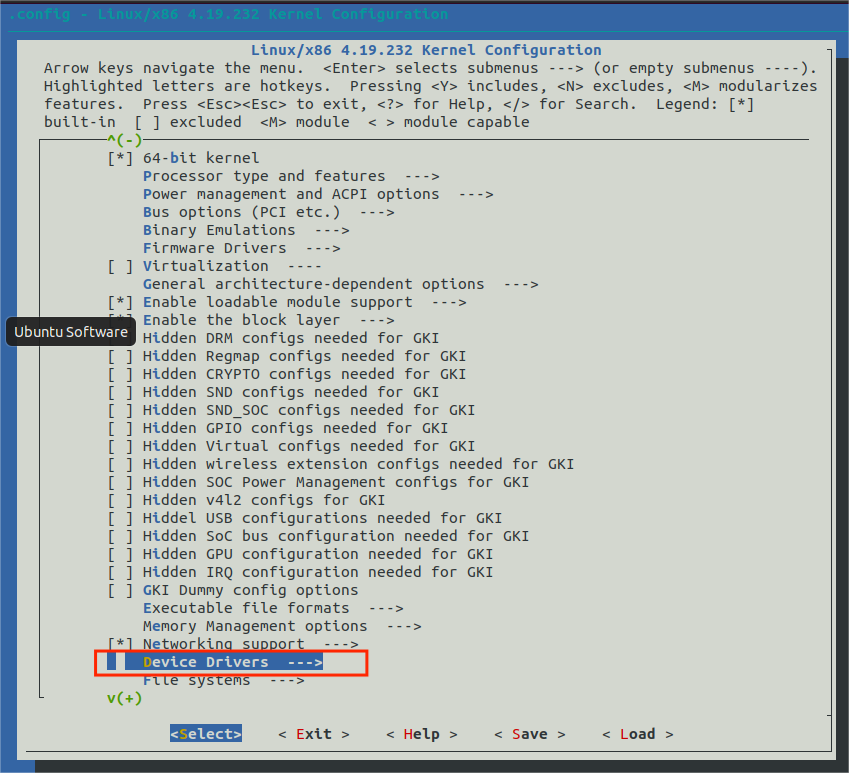
我们看看菜单：menuconfig会有什么变化：在内核目录下输入：

make menuconfig

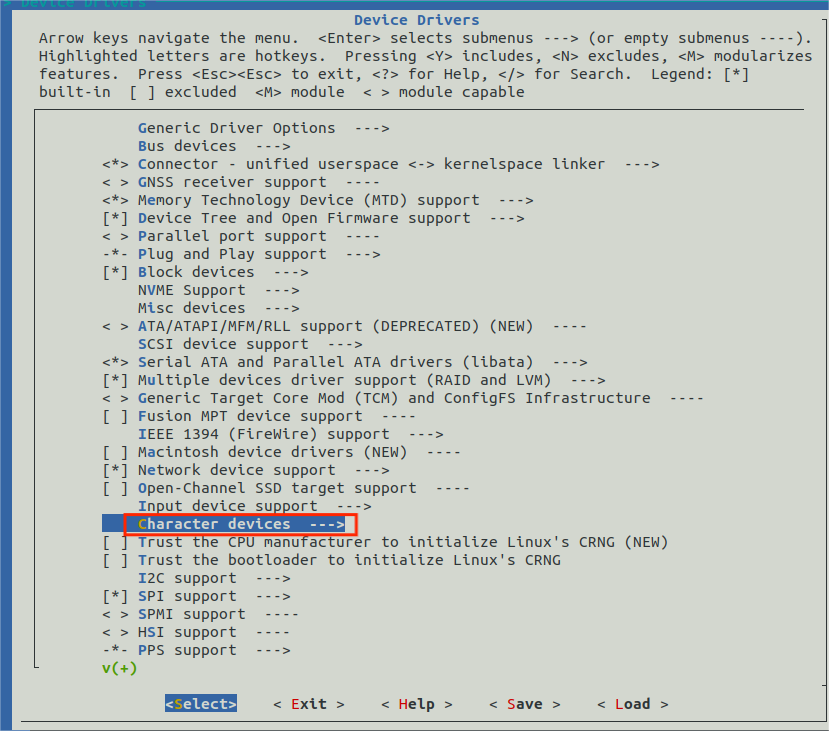
(注意，我们使用的是arm64平台)



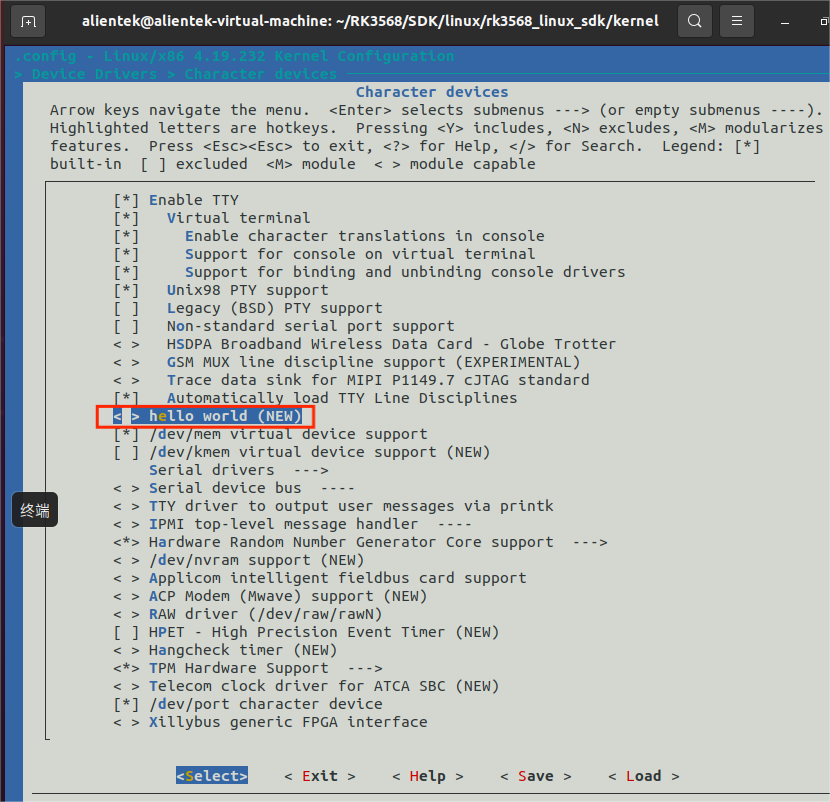
1. 我们进入到device drivers下面



1. 然后进入到Character devices选项下面：



（3）我们就可以看到我们新增加的Hello World选项。我们可以敲击空格改写状态：

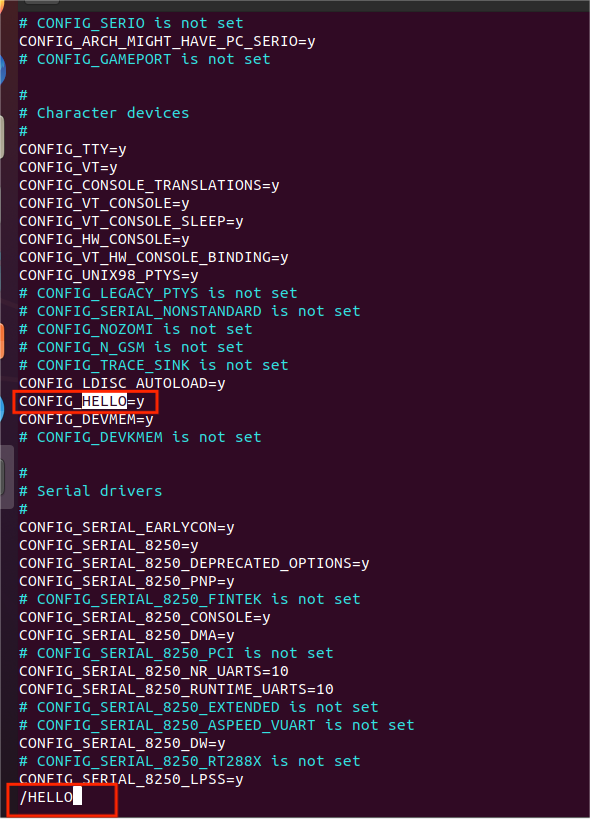


IMG_256

IMG_256

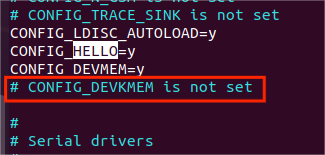
（4）我们选择编译进内核（<\*>版本）：然后退出保存。

（5）此时就可以直接编译了，这里为了保险起见，我们先看一下.config文件，这里我们直接搜索HELLO，直接输入“/HELLO”。可以看到：



这里指等于y，y就是将该模块编译进入内核。

1. 我们之前提到，可以直接修改.config文件：



由图可见：框内的模块没有设置，我们修改为下面的形式：

CONFIG\_DEVKMEM=y,如图：

IMG_256

这种当然也可以，但是可能会失败。所以这种方法可以用，但不推荐。

我们撤销回去：退出编辑模式后，按u撤销。

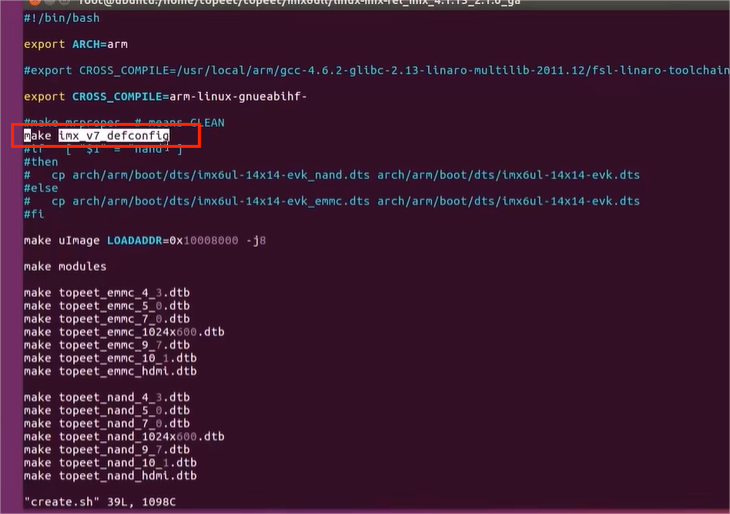
——————————————————————————————————

此处开始make.sh以及arch/arm64/configs的配置

9.我们查看完.config之后，还不能直接编译。还需要查看kernel目录下的编译脚本:creat.sh脚本，（我们可能是make.sh）

/\*

他的creat.sh脚本:



其中的这一句话是内核的默认用法：也就是说在编译的时候，他会自己去找arch/arm这个下面的imx\_v7这个配置文件，作为我们的.config来编译我们的内核，那么他修改了这个.config文件，在.config文件里面有了Hello。但是编译的时候还不是用的.config来编译，而是用的arch/arm/configs/imx\_v7\_defconfig文件来编译的。所以还需要改一下：

1. 先在内核目录下执行make distclean 清除所有的编译文件

\*/

/\*

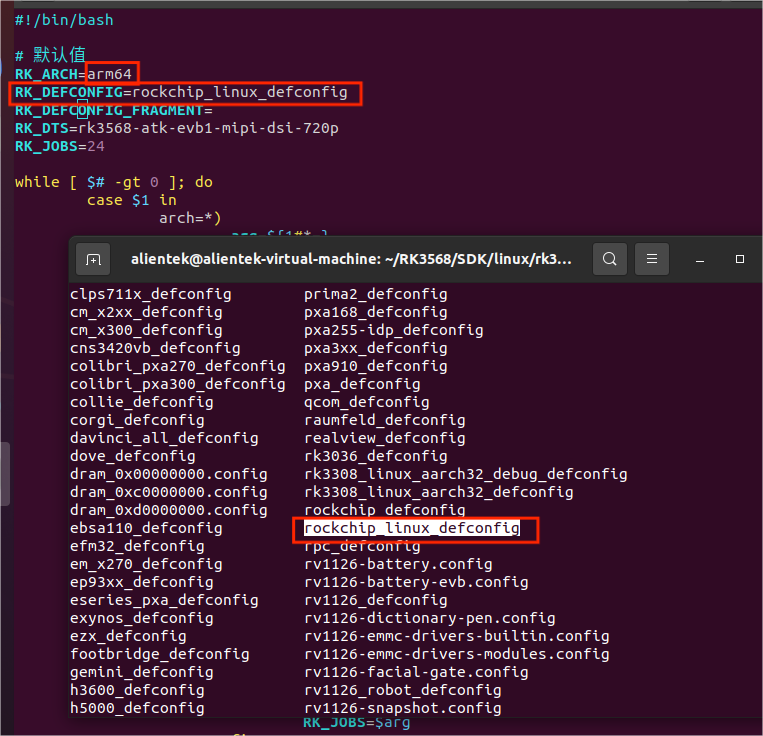
我的编译脚本：

后面的是：

~/RK3568/SDK/linux/rk3568\_linux\_sdk/kernel路径下的make.sh脚本

前面的是：

~/RK3568/SDK/linux/rk3568\_linux\_sdk/kernel/arch/arm/configs下的配置文件。



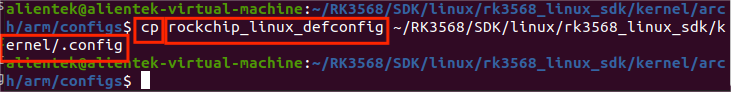
由此可知，它实际上用的是rockchip\_linux\_defconfig文件来配置内核，并且选择的平台是arm64，所以不能使用arm平台下的rockchip\_linux\_defconfig

/\*

这里是错误版本，错误地使用了arm平台，而不是arm64,但方法都一样，后续需要将arm平台都改为arm64目录即可

1. 首先将rockchip\_linux\_defconfig拷贝到内核目录下并命名为.config：

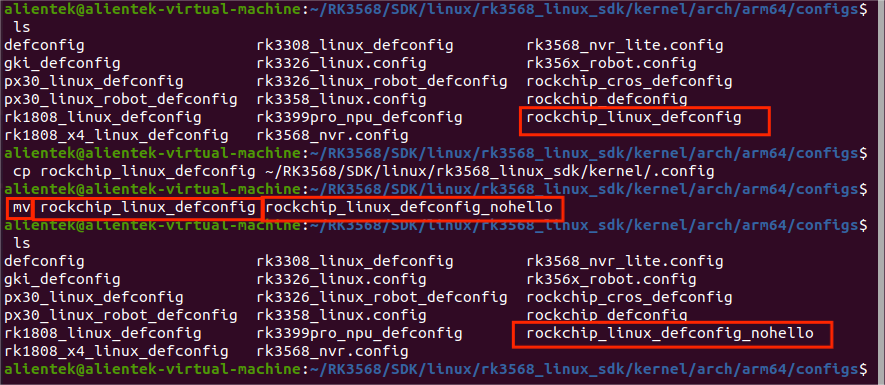
cp rockchip\_linux\_defconfig ~/RK3568/SDK/linux/rk3568\_linux\_sdk/kernel/.config

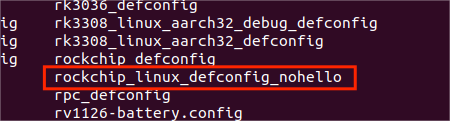


1. 执行make menuconfig，并选择hello world项将其编译进内核。操作同前面
2. 对rockchip\_linux\_defconfig进行备份，此处备份为

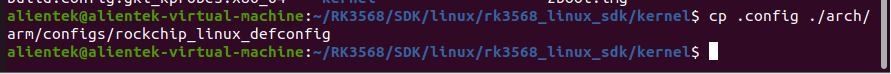
rockchip\_linux\_defconfig\_nohello：

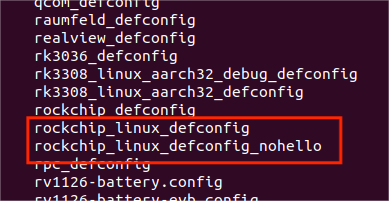
mv rockchip\_linux\_defconfig rockchip\_linux\_defconfig\_nohello



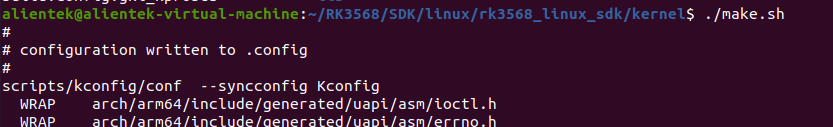


1. 将kernel目录下的.config文件拷贝到/kernel/arch/arm/configs下，并改名为rockchip\_linux\_defconfig





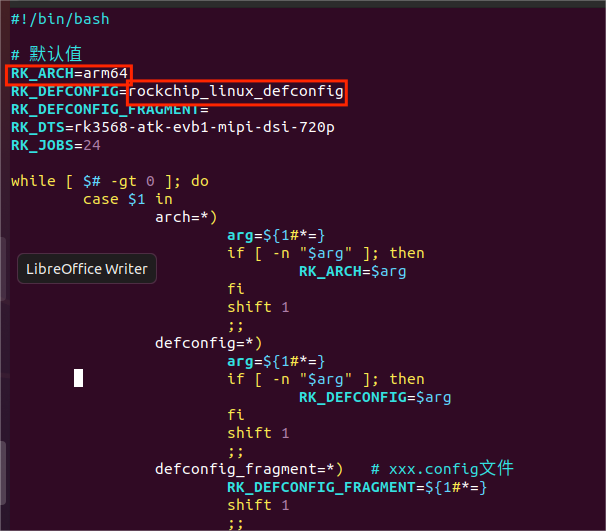
1. 再回到kernel目录下，再编译再运行make.sh脚本文件,那么它就是用的配置好的，有hello的rockchip\_linux\_defconfig文件。
2. 改写好了之后，就可以直接运行脚本：./make.sh



1. 编译完成之后，有没有编译成功还不知道，检验编译成没成功有两种方法：
   1. 直接到开发板上运行，看看有没有加载这个驱动
   2. 去drivers/char/hello/目录下，看有没有把.c文件编译成.o文件。如果没有.o文件一般就是哪一步做错了。

/\*

这里编译失败了，没有生成helloworld.o文件，查找了一圈，看到kernel目录下的make.sh脚本：

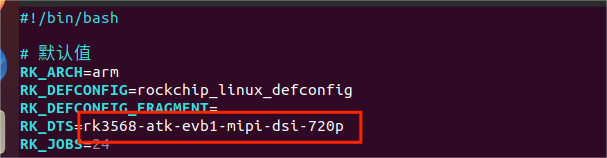


这里可以看到ARCH选择的是arm64,我们修改的是arm，所以猜测可能是编译的平台问题。因此将其修改成arm进行尝试。

编译失败了：



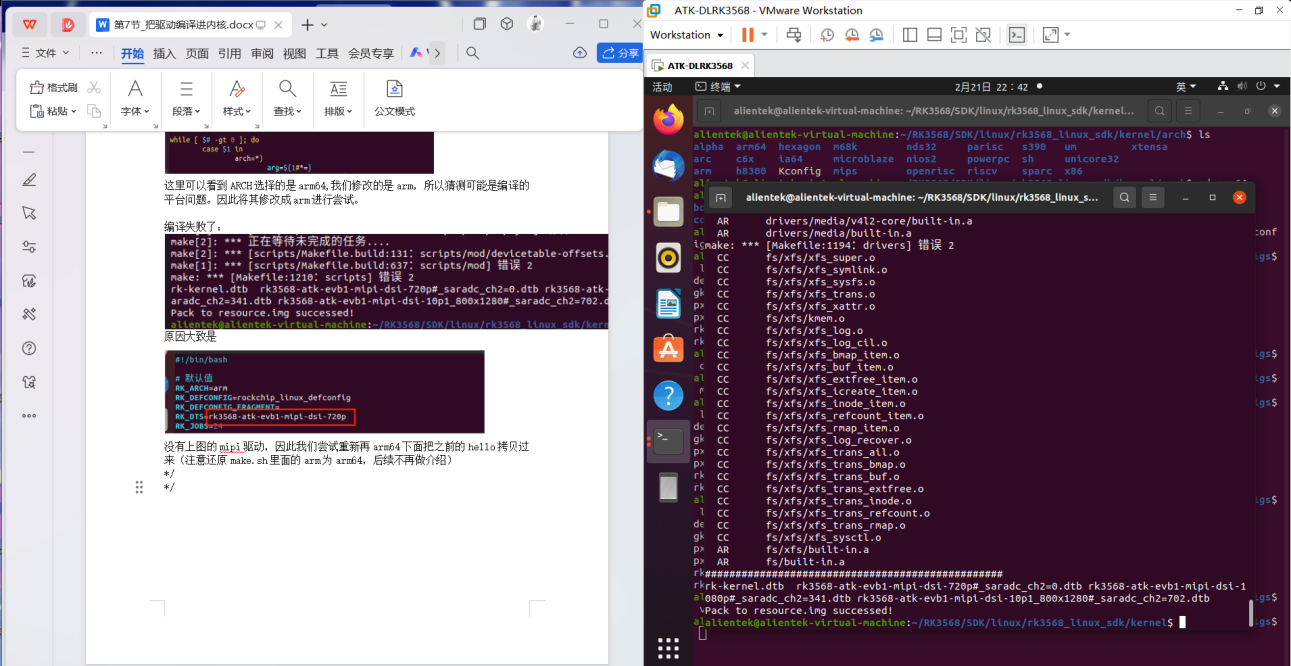
原因大致是



没有上图的mipi驱动，因此我们尝试重新再arm64下面把之前的hello拷贝过来（注意还原make.sh里面的arm为arm64，后续不再做介绍）

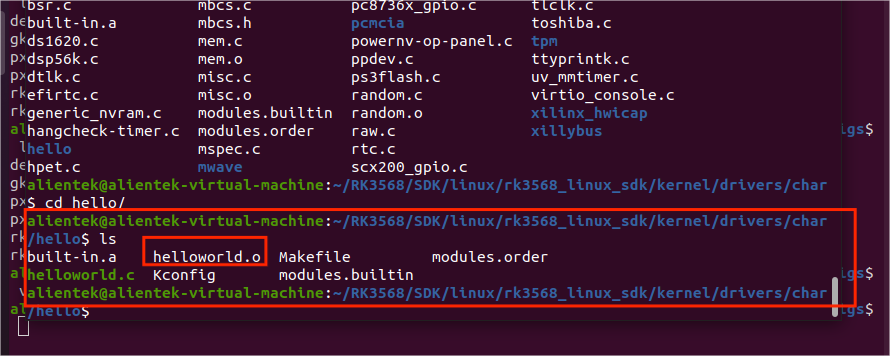
\*/

\*/



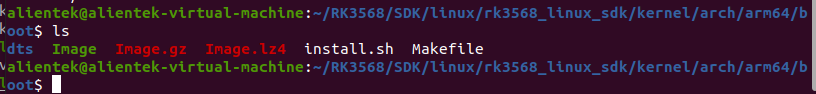
接上面，修改了arm64之后再重新编译，此时显示编译完成。





此时就会成功生成helloworld.o文件

同时在kernel/arch/arm64/boot目录下也会生成对应的目标镜像：



Kernel的编译方法：见"C:\Users\zhongqing\Desktop\LInux开发笔记\Linux\_工具配置.doc"的 6.2 kernel内核开发。