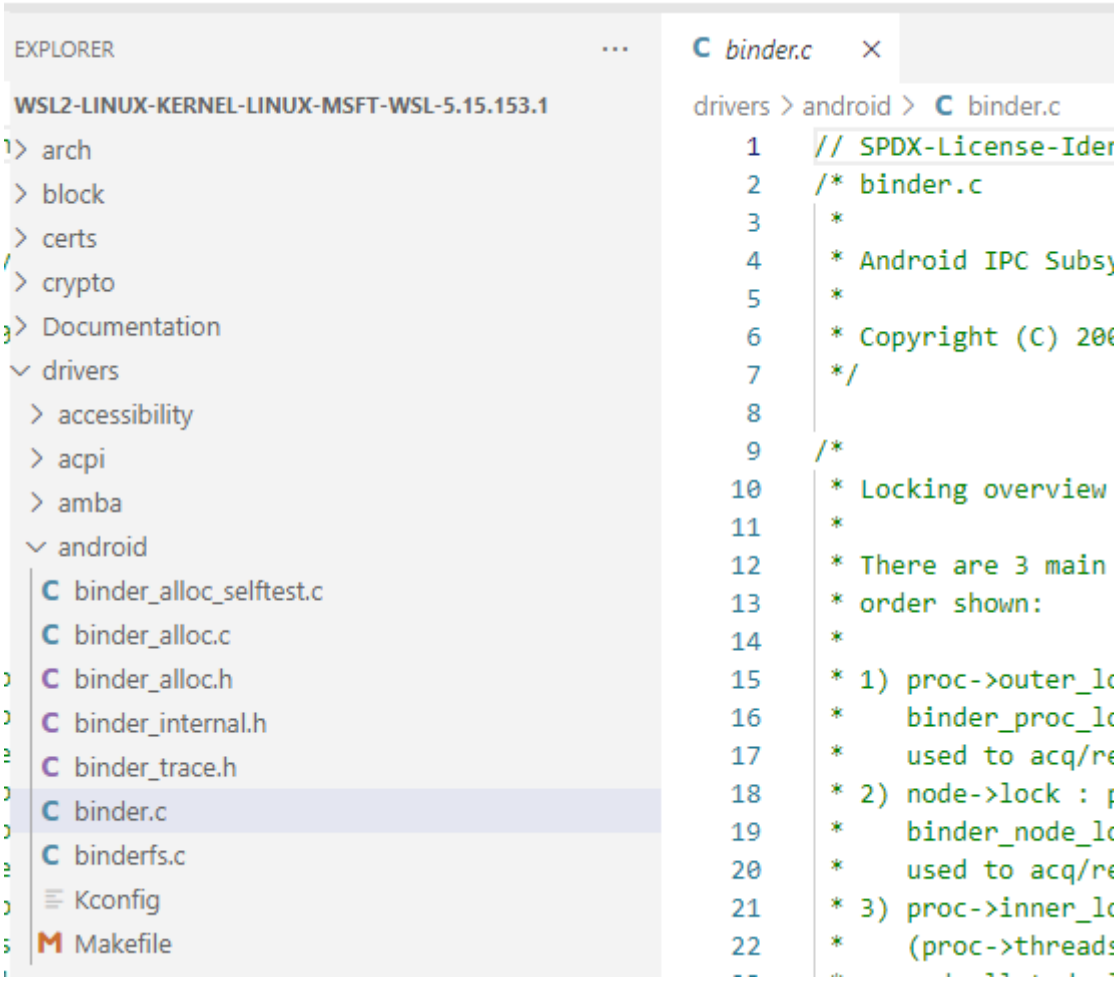
WSL替换自编译内核 & 插入自编译模块

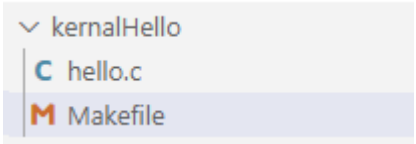
1、背景

目前Android的Binder驱动，是在Linux的源码模块中的，

drivers\android\



假如我们希望对该代码进行修改做一些测试，那么会依赖一些linux内核源码的头文件，用以下代码为例：



-TestModule

-hello.c

-Makefile

以上为整个编译的目录TestModule，该目录下只有两一个文件，一个为hello.c，一个为Makefile，对应的代码为：

helllo.c:

```c
#include<linux/module.h>
#include<linux/kernel.h>

MODULE_LICENSE("GPL");

static int __init hello_mod_init(void) {
    printk(KERN_ALERT "Hello world from kernel!! \n");
    return 0;
}

static void __exit hello_mod_exit(void) {
    printk(KERN_ALERT "Exiting hello world module from kernel !!!\n");
}

module_init(hello_mod_init);
module_exit(hello_mod_exit);
```

Makefile:

```
obj-m +=hello.o
KDIR:=/lib/modules/$(shell uname -r)/build
PWD:=$(shell pwd)
default:
    $(MAKE) -C $(KDIR) M=$(PWD) modules

clean:
    $(MAKE) -C $(KDIR) M=$(PWD) clean
```

可以看到这个代码中，依赖了linux/module.h 跟linux/kernel.h， 我们的目标是使用make构建该hello.c文件，生成对应的内核模块，生成后的内核模块输出为.ko后缀的文件吗，这个例子中就是hello.ko。

生成内核成功后，我们通过`sudo insmod hello.ko`将该模块装载到内核当中，同样也可以通过

`rmmod hello`将hello模块从内核模块中移除。

2、编译自己的WSL内核

2.1、下载源码

首先 使用`uname -r`查看你当前的wsl版本

```
root@DESKTOP-JOH1FN4:/# uname -r
5.15.153.1-microsoft-standard-WSL2
```

以上是我的输出，接着去github上下载对应的源码

https://github.com/microsoft/WSL2-Linux-Kernel/releases

在该链接中找到你对应的版本，并下载对应的 Source code(tar.gz)

下载好之后，渠道WSL根目录，在home目录下创建一个文件夹 (不要使用mnt目录，不然会有各种各样奇怪的问题)，把对应的tar.gz压缩包 通过cp命令copy过来，进行解压

`tar -zxvf WSL2-Linux-Kernel-linux-msft-wsl-5.15.153.1.tar.gz`

解压后，我的目录如下：

```
root@DESKTOP-JOH1FN4:/home/linuxkernel# ls
WSL2-Linux-Kernel-linux-msft-wsl-5.15.153.1
```

2.2、编译内核源码

首先使用apt下载一个前置的工具包：

```
sudo apt install libelf-dev build-essential pkg-config bison flex libssl-dev bc dwarves
```

这里的工具包，还有make， gcc等都要确保都下载好，如果一些下载不下来的，使用 `sudo apt update` 更新后重试一下，如果还有问题，就去找下别的博客，看看解决方案，目前我试下来这些工具包基本都没有什么下载的异常，只有dwarves 跟 bison，需要先执行下 `sudo apt update`

接下来进入到WSL2-Linux-Kernel-linux-msft-wsl-5.15.153.1的源码根目录

执行以下命令

```
cp Microsoft/config-wsl .config
make -j$(nproc)
make modules -j$(nproc)
make modules_install -j$(nproc)
make install -j$(nproc)
```

这里的nproc代表编译的最大线程使用数，不指定默认一个线程，我是指定8来编译的

```
cp Microsoft/config-wsl .config
make -j8
make modules -j8
make modules_install -j8
make install -j8
```

这里每一步的执行还是很顺利的，linux的代码并不多，编译起来也基本没有奇奇怪怪的编译问题

这里的第一条命令 `make -j8` 执行完以后，你可以看到最后成功的输出为：

```
  OBJCOPY arch/x86/boot/setup.bin
  BUILD   arch/x86/boot/bzImage
Kernel: arch/x86/boot/bzImage is ready  (#4)
```

这里的 `Kernel: arch/x86/boot/bzImage` 就是生成的新的内核镜像文件。 我们可以先将内核镜像文件复制到桌面或者其他文件夹中保存一下：

我先把这个文件保存在我的E盘下 `E:\UbuntuWSL\bzImage`

剩下的三条命令执行完以后，可以去`/lib/modules`目录下check一下

```
root@DESKTOP-JOH1FN4:/lib/modules# ls
5.15.153.1-microsoft-standard-WSL2
```

可以看到会存在一个目录，进入5.15.153.1-microsoft-standard-WSL2

```
root@DESKTOP-JOH1FN4:/# cd lib/modules
root@DESKTOP-JOH1FN4:/lib/modules# ls
5.15.153.1-microsoft-standard-WSL2
root@DESKTOP-JOH1FN4:/lib/modules# cd 5.15.153.1-microsoft-standard-WSL2/
root@DESKTOP-JOH1FN4:/lib/modules/5.15.153.1-microsoft-standard-WSL2# ll
total 356
drwxr-xr-x 3 root root   4096 Aug  4 15:29 ./
drwxr-xr-x 3 root root   4096 Aug  4 15:29 ../
lrwxrwxrwx 1 root root     61 Aug  4 15:29 build -> /home/linuxkernel/WSL2-Linux-Kernel-linux-msft-wsl-5.15.153.1/
drwxr-xr-x 4 root root   4096 Aug  4 15:29 kernel/
-rw-r--r-- 1 root root  58189 Aug  4 15:29 modules.alias
-rw-r--r-- 1 root root  59397 Aug  4 15:29 modules.alias.bin
-rw-r--r-- 1 root root  15144 Aug  4 15:29 modules.builtin
-rw-r--r-- 1 root root  44824 Aug  4 15:29 modules.builtin.alias.bin
-rw-r--r-- 1 root root  17273 Aug  4 15:29 modules.builtin.bin
-rw-r--r-- 1 root root 111306 Aug  4 15:29 modules.builtin.modinfo
-rw-r--r-- 1 root root    619 Aug  4 15:29 modules.dep
-rw-r--r-- 1 root root   1150 Aug  4 15:29 modules.dep.bin
-rw-r--r-- 1 root root      0 Aug  4 15:29 modules.devname
-rw-r--r-- 1 root root    434 Aug  4 15:29 modules.order
-rw-r--r-- 1 root root     55 Aug  4 15:29 modules.softdep
-rw-r--r-- 1 root root   1286 Aug  4 15:29 modules.symbols
-rw-r--r-- 1 root root   1294 Aug  4 15:29 modules.symbols.bin
lrwxrwxrwx 1 root root     61 Aug  4 15:29 source -> /home/linuxkernel/WSL2-Linux-Kernel-linux-msft-wsl-5.15.153.1/
```

当你看到

```
lrwxrwxrwx 1 root root     61 Aug  4 15:29 build -> /home/linuxkernel/WSL2-Linux-Kernel-linux-msft-wsl-5.15.153.1/
lrwxrwxrwx 1 root root     61 Aug  4 15:29 source -> /home/linuxkernel/WSL2-Linux-Kernel-linux-msft-wsl-5.15.153.1/
```

这两个信息，你的内核模块编译跟装载基本没问题了，接下来把内核代码的头文件依赖进来，执行以下命令：

```
sudo make headers_install ARCH=x86 INSTALL_HDR_PATH=/usr
```

执行完这条命令后，进入到`/usr/include`目录我们可以看到一些内核源码的头文件

```
root@DESKTOP-JOH1FN4:/usr/include# ll
total 1616
drwxr-xr-x 38 root root   4096 Aug  4 11:59 ./
drwxr-xr-x 14 root root   4096 Apr 23  2020 ../
-rw-r--r--  1 root root   6893 Aug  7  2018 FlexLexer.h
```

```
-rw-r--r--  1 root root   3449 Aug  4 11:59 Makefile
drwxr-xr-x  2 root root   4096 Feb  8  2020 X11/
-rw-r--r--  1 root root   7457 Apr 15  2020 aio.h
-rw-r--r--  1 root root   2032 Apr 15  2020 aliases.h
-rw-r--r--  1 root root   1204 Apr 15  2020 alloca.h
-rw-r--r--  1 root root   1731 Apr 15  2020 ar.h
-rw-r--r--  1 root root  25473 Apr 15  2020 argp.h
-rw-r--r--  1 root root   6051 Apr 15  2020 argz.h
... 头文件比较多，省略展示
```

2.3、替换WSL的内核为我们编译的内核

回到刚刚我们执行 `make -j8` 后，我们将输出的镜像文件 `Kernel: arch/x86/boot/bzImage` 复制到了 `E:\UbuntuWSL\` 目录下， 下面我们需要修改一下配置，让WSL启动时使用我们编译的内核

在C盘的用户目录下 C:\Users\{你的用户名}

新建一个 .wslconfig文件

在文件中编辑以下内容：

```
[wsl2]
kernel=E:\\UbuntuWSL\\bzImage
```

这里就是一个配置，把内核指向我们编译的模块。

这个防范是微软推荐的，具体可查看： https://learn.microsoft.com/en-us/windows/wsl/wsl-config#configure-global-options-with-wslconfig

配置完成后需要重启wsl， 通过 `wsl shudown` 来完成，但我这边shutdown总是失败，没办只能通过重启电脑来完成。

重启wsl后，一切顺利的话，你已经使用了我们自己编译的内核。

该怎么确认替换成功呢?
可以通过 `uname -r` 查看版本，如果你编译的版本和你原本wsl的版本一致的话，可能看不出来，如果想做出区分，需要在编译前修改源码目录下的 `.config` 文件，里面可以把 `EXTRAVERSION` 字段换成你希望标记的名字，这样可以通过 `uname -r` 命令反馈出来。

不过你不想重新把前面流程走一遍的话可以先不管，可以继续看后编译自己内核模块，然后插入，假如是用的官方内核的话，不意外你会出现各种问题，比如我在没切换版本，插入内核模块时，会遇到以下错误：（dmesg是查看系统的日志的命令）

```
root@DESKTOP-JOH1FN4:/mnt/e/OsTestCode/kernalHello# sudo insmod hello.ko
insmod: ERROR: could not insert module hello.ko: Invalid parameters
root@DESKTOP-JOH1FN4:/mnt/e/OsTestCode/kernalHello# sudo dmesg -c
[11386.833205] BPF:[133426] FWD
[11386.833418] BPF:struct
[11386.833517] BPF:
[11386.833611] BPF:Invalid name
[11386.833752] BPF:

[11386.833894] failed to validate module [hello] BTF: -22
```

这个错误也没显示具体的信息，在我没有切换内核实现时一直发生。

3、编译自定义内核模块

回到这段代码：helllo.c：

```c
#include<linux/module.h>
#include<linux/kernel.h>

MODULE_LICENSE("GPL");

static int __init hello_mod_init(void) {
    printk(KERN_ALERT "Hello world from kernel!! \n");
    return 0;
}

static void __exit hello_mod_exit(void) {
    printk(KERN_ALERT "Exiting hello world module from kernel !!!\n");
}

module_init(hello_mod_init);
module_exit(hello_mod_exit);
```

Makefile:

```
#指定输出，名称要问c文件相同，后缀为.o
obj-m +=hello.o
KDIR:=/lib/modules/$(shell uname -r)/build
PWD:=$(shell pwd)
default:
    $(MAKE) -C $(KDIR) M=$(PWD) modules

clean:
    $(MAKE) -C $(KDIR) M=$(PWD) clean
```

可以看到在make文件中，指定的KDIR为：`/lib/modules/$(shell uname -r)/build`

我们uname -r是我们的内核版本，比如为 `WSL2-Linux-Kernel-linux-msft-wsl-5.15.153.1`

那么KDIR为：`/lib/modules/5.15.153.1-microsoft-standard-WSL2/build`

我们在make编译前，先去检查下这个目录是否存在，我这边遇到一个问题是，我重启wsl时，虽然内核版本替换了，但是 `/lib/modules/` 变成了一个空的文件夹，不过问题不大，我们不需要重新编，只需要重新渠道源码目录，执行以下命令做一些安装即可：

```
sudo make modules_install -j8
make install -j8
```

执行以上命令后，再进到 `/lib/modules` 目录下check下，编译依赖的kdir目录是否存在。

解下来, 进到自定模块的源代码目录开始编译：

```
root@DESKTOP-JOH1FN4:/mnt/e/OsTestCode/kernalHello# make
root@DESKTOP-JOH1FN4:/mnt/e/OsTestCode/kernalHello# sudo insmod hello.ko
```

装载后可以使用 `dmesg -c` 查看有没有预期的输出，实际执行结果

```
root@DESKTOP-JOH1FN4:/mnt/e/OsTestCode/kernalHello# ls
Makefile  hello.c

root@DESKTOP-JOH1FN4:/mnt/e/OsTestCode/kernalHello# make
make -C /lib/modules/5.15.153.1-microsoft-standard-WSL2/build
M=/mnt/e/OsTestCode/kernalHello modules
make[1]: Entering directory '/home/linuxkernel/WSL2-Linux-Kernel-linux-msft-
wsl-5.15.153.1'
  CC [M]  /mnt/e/OsTestCode/kernalHello/hello.o
  MODPOST /mnt/e/OsTestCode/kernalHello/Module.symvers
  CC [M]  /mnt/e/OsTestCode/kernalHello/hello.mod.o
  LD [M]  /mnt/e/OsTestCode/kernalHello/hello.ko
  BTF [M] /mnt/e/OsTestCode/kernalHello/hello.ko
make[1]: Leaving directory '/home/linuxkernel/WSL2-Linux-Kernel-linux-msft-
wsl-5.15.153.1'


root@DESKTOP-JOH1FN4:/mnt/e/OsTestCode/kernalHello# sudo insmod hello.ko

root@DESKTOP-JOH1FN4:/mnt/e/OsTestCode/kernalHello# dmesg -c
[  655.379081] Hello world from kernel!!
```

可以看到自定义内核模块被正常加载了

另外 `lsmod` 也可以看到被装载的模块

```
root@DESKTOP-JOH1FN4:/mnt/e/OsTestCode/kernalHello# lsmod
Module                  Size  Used by
hello                  16384  0
```

可以看到hello可以正常装载。

接下来测试卸载模块：

```
root@DESKTOP-JOH1FN4:/mnt/e/OsTestCode/kernalHello# sudo rmmod hello
root@DESKTOP-JOH1FN4:/mnt/e/OsTestCode/kernalHello# dmesg -c
```

实际执行结果为：

```
root@DESKTOP-JOH1FN4:/mnt/e/OsTestCode/kernalHello# sudo rmmod hello
root@DESKTOP-JOH1FN4:/mnt/e/OsTestCode/kernalHello# dmesg -c
[  677.749768] Exiting hello world module from kernel !!!
```

ok，到此为止，我们完成了自定义内核的编译与加载， 可以在自定义内核中可以直接调用内核的一些源码头文件暴露的接口，方便了我们进行各种测试。

后续使用注意下这个问题：

> **我们在make编译前，先去检查下这个目录是否存在，我这边遇到一个问题是，我重启wsl时，虽然内核版本替换了，但是 `/lib/modules/` 变成了一个空的文件夹，不过问题不大，我们不需要重新编，只需要重新渠道源码目录，执行以下命令做一些安装即可：**
>
> ```
> sudo make modules_install -j8
> make install -j8
> ```
>
> 执行以上命令后，再进到 `/lib/modules` 目录下check下，编译依赖的kdir目录是否存在。

这应该我某些不当的操作引起的，等后续有空再研究，但不影响使用，只是可能在使用前执行下以上两个命令即可。