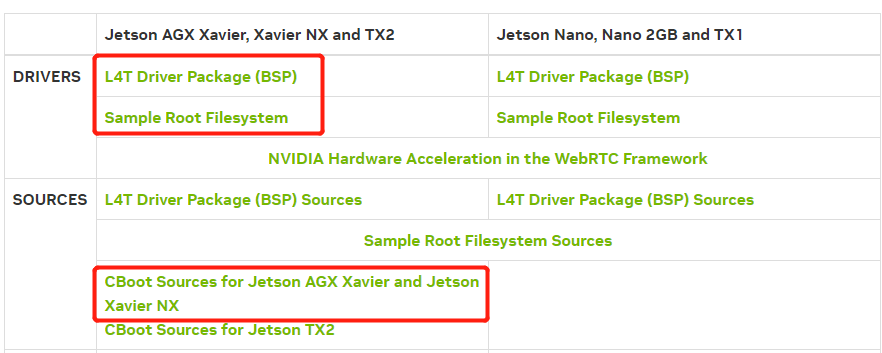
**Cboot加固方案设计与操作流程**

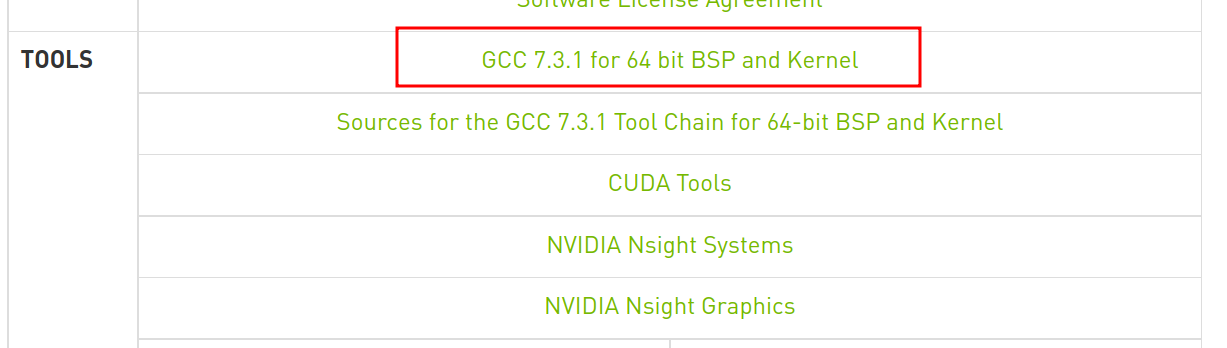
1. **固件烧录流程**
2. **全固件烧录**

（1）下载Jetson开发工具包，从下列网址中选择下载合适版本的Jetson Linux发行包和示例文件系统：<https://developer.nvidia.com/linux-tegra>

如图，下载Jetson Linux R32.4.4版本：







下载后尽量将下列四个压缩包（Tegra186\_Linux\_R32.4.4\_aarch64.tbz2、

Tegra\_Linux\_Sample-Root-Filesystem\_R32.7.3\_aarch64.tbz2.part、

gcc-linaro-7.3.1-2018.05-x86\_64\_aarch64-linux-gnu.tar.xz、cboot\_src\_t19x.tbz2）置于同一目录下，方便接下来的操作：



（2）使用以下指令来解压缩文件并组装rootfs：

I.使用以下指令解压缩Tegra186\_Linux\_R32.4.4\_aarch64.tbz2压缩包，获得Linux\_for\_Tegra文件夹：

$ tar xf ${L4T\_RELEASE\_PACKAGE}

${L4T\_RELEASE\_PACKAGE}为Tegra186\_Linux\_R32.4.4\_aarch64.tbz2压缩包的名称，不同版本的压缩包名称可能不一致，不过解压后都将得到Linux\_for\_Tegra文件夹。

II.进入Linux\_for\_Tegra文件夹下的rootfs文件夹，发现rootfs文件夹为空，使用以下指令将示例文件系统压缩包（Tegra\_Linux\_Sample-Root-Filesystem\_R32.7.3\_aarch64.tbz2.part）解压到rootfs文件夹：

$ cd Linux\_for\_Tegra/rootfs/

$ sudo tar xpf ../../${SAMPLE\_FS\_PACKAGE}

../../${SAMPLE\_FS\_PACKAGE}为示例文件系统压缩包的路径，由于示例文件系统压缩包超过1G，解压缩需要一点时间。

III.返回上一级Linux\_for\_Tegra文件夹下，使用下列指令执行apply\_binaries.sh脚本：

$ cd ..

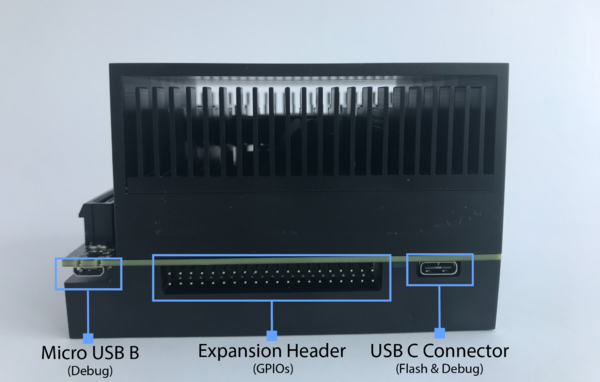
$ sudo ./apply\_binaries.sh

此步骤非常重要，目的是链接填充各文件夹下的文件，如果忽略此步骤，后续将报错。

（3）使Jetson开发套件置于强制恢复模式，对于Jetson AGX Xavier系列，使其进入强制恢复模式的具体操作如下：

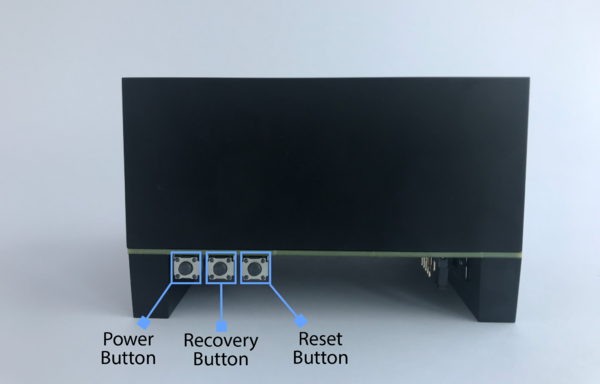
I.关闭AGX电源；

II.接好3个按键旁边的TypeC口（USB C Connector）到PC端，如图所示：



III.AGX接上电源；

IV.按下中间的Recovery按键不放，如下图所示：

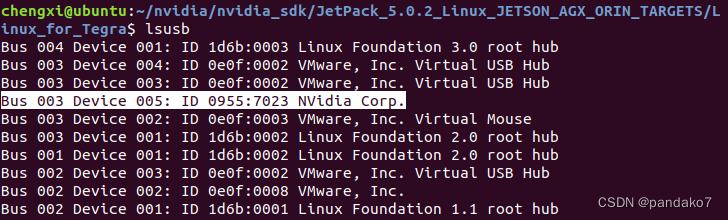


V.按一下电源按键，灯亮；

VI.等待5秒钟后，松开中间的Recovery按键，AGX进入强制恢复模式。

如果AGX设备处于开机状态，想要进入强制恢复模式，可先按住Recovery键，再按一下Reset键，等待大约5秒后松开Recovery键，即可进入强制恢复模式。

在Ubuntu终端上执行lsusb指令，如果发现NVidia Corp标识，如下图所示，说明设备已进入恢复模式。



（4）在PC主机端执行下列指令，通过脚本将目标设备刷机：

sudo ./flash.sh jetson-agx-xavier-devkit mmcblk0p1

该指令将编译整个Linux\_for\_Tegra文件夹并将rootfs文件夹整个烧录进AGX设备，需要一些时间，**如果想要只烧录引导加载程序（BootLoader）而不烧录内核镜像文件等，可使用以下指令，该指令可用来只替换Cboot启动程序，只替换Cboot启动程序的具体操作见下：**

sudo ./flash.sh -r -k cpu-bootloader jetson-agx-xavier-devkit mmcblk0p1

1. **只烧录Cboot**
   1. **Cboot编译**

由于Cboot加固项目提供修改过后的Cboot源代码，所以不需要从官网下载源代码，想要编译Cboot加固项目源代码，可以直接从第2步开始配置环境变量，官方编译Cboot源代码的流程如下：

1. 下载Cboot源代码程序，下载地址链接为：

https://developer.nvidia.com/embedded/linux-tegra-r3244；

（2）使用下列命令提取CBoot独立源：

$ mkdir cboot

$ tar -xjf cboot\_src\_t194.tbz2 -C cboot

$ cd cboot

1. 至NVIDIA官方网址下载编译需要的Toolchain，NVIDIA官方网址如下：

https://developer.nvidia.com/embedded/jetson-linux-archive

下载后，使用以下命令进行解压缩：

$ tar -xvpJf gcc-linaro-7.3.1-2018.05-x86\_64\_aarch64-linux-gnu.tar.xz

（4）使用下列命令配置环境参数：

$ export CROSS\_COMPILE=<your\_64-bit\_ARM\_toolchain>

$ export TEGRA\_TOP=$PWD

$ export TOP=$PWD

其中，<your\_64-bit\_ARM\_toolchain>是Toolchain解压缩后的位置，例如：

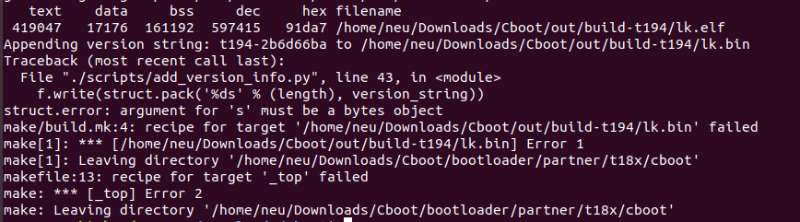
$export CROSS\_COMPILE=gcc-linaro-7.3.1-2018.05-x86\_64\_aarch64-linux-gnu/bin/aarch64-linux-gnu-

（5）使用下列命令编译Cboot代码程序：

$ make -C ./bootloader/partner/t18x/cboot PROJECT=t194

TOOLCHAIN\_PREFIX="${CROSS\_COMPILE}" DEBUG=2 BUILDROOT="${PWD}"/out NV\_TARGET\_BOARD=t194ref NV\_BUILD\_SYSTEM\_TYPE=l4t NOECHO=@

编译结束可能出现下列情况：



输入指令重新编译一次，错误消失：



出现上面情况可能是python版本的问题，不影响Cboot开发。

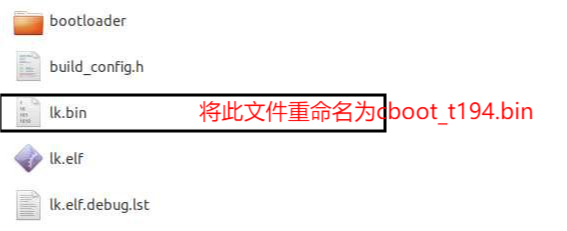
个人在ubuntu18版本下使用python3.6开发Cboot，在ubuntu20版本下使用python3.8开发Cboot，都能编译成功，最多出现上面情况，然而在ubuntu22版本下使用python3.10开发Cboot，无法编译成功，经调查，NVIDIA官方曾在官网论坛谈及Cboot是在ubuntu18版本下开发，不建议在最新版本ubuntu下编译Cboot。

编译完成后，将在Cboot文件夹下产生一个out文件夹，如图所示。



* 1. **Cboot烧写**

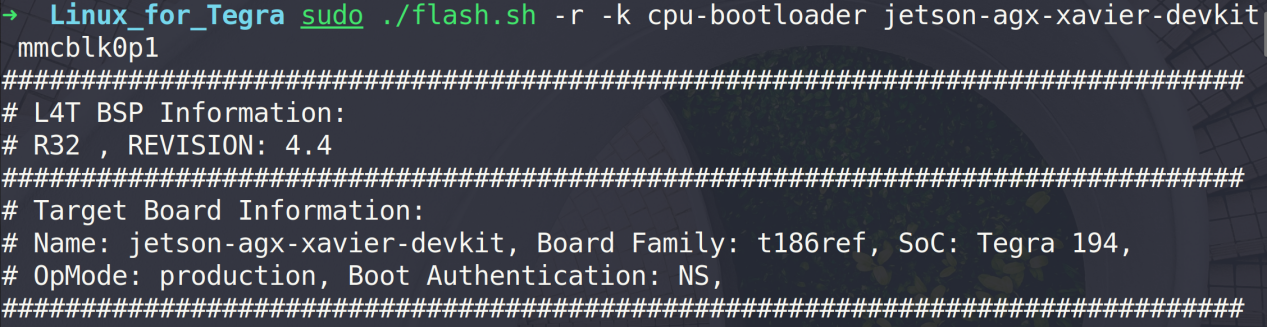
（1）将cboot/out/build-t194下的lk.bin文件重命名为cboot\_t194.bin，如图所示，并复制到Linux\_for\_Tegra/bootloader文件夹下取代原来的cboot\_t194.bin文件。

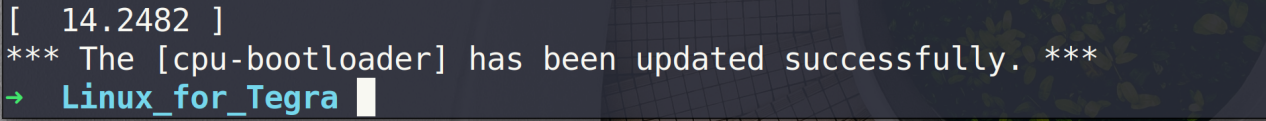


（2）先使AGX进入强制恢复模式，再使用下列指令只替换BootLoader：

sudo ./flash.sh -r -k cpu-bootloader jetson-agx-xavier-devkit mmcblk0p1

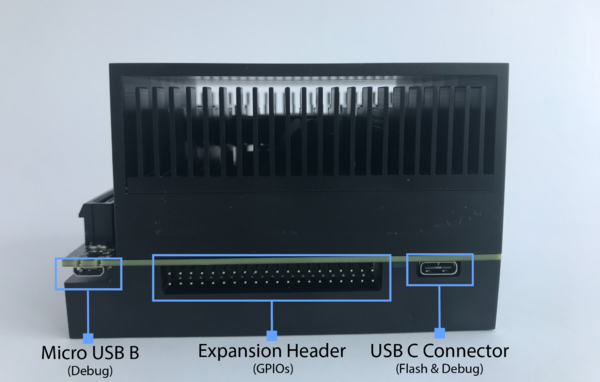
替换成功后如图所示：





1. **串行控制台（Serial console）安装**

NVIDIA Jetson系列提供一个USB连接器，用于使用串行控制台进行调试，调试端口位置可以在开发板的一侧找到，如图所示。



使用micro USB电缆将此端口连接到运行Ubuntu的PC端后，将自动在PC端中注册四个串行端口。假设连接之前不存在其他/dev/ttyUSBn，则将为这些端口分配设备/dev/ttyUSB0、/dev/ttyUSB1、/dev/ttyUSB2和/dev/ttyUSB3，调试端口通常是最后一个位置，即/dev/ttyUSB3。

使用下列指令安装minicom：

$ apt install minicom

使用下列指令打开minicom：

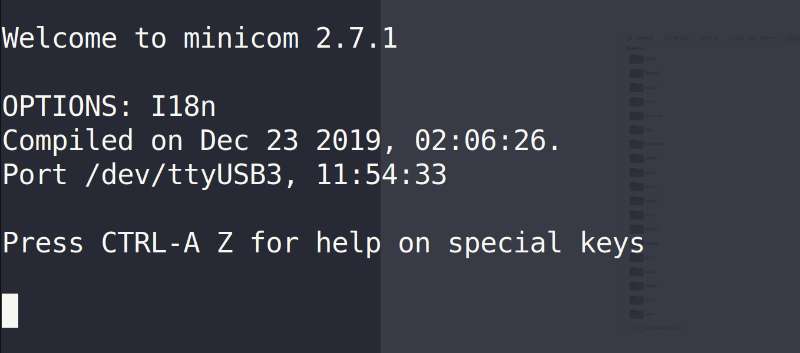
$ minicom -D /dev/ttyUSB3 -8 -b 115200

如果提示无权限，可以使用下列指令进入超级用户：

$ su

如果执行$ su指令后仍然提示无权限，可先执行$ sudo passwd root指令指定超级用户密码，接着执行$ su指令，输入刚刚设置的密码便可以进入超级用户。

Minicom界面如图所示：



1. **Cboot加固方案设计及实现**
2. **Cboot加固方案设计**

Cboot实现Kernel镜像加固的具体流程如下：

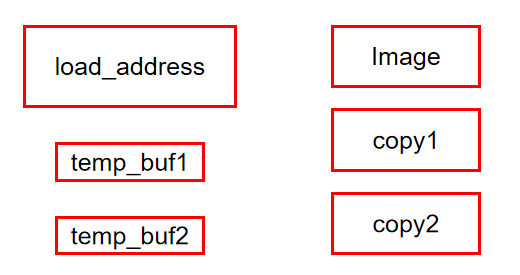
（1）首先在主机的Linux\_for\_Tegra/rootfs/boot文件夹下准备三份操作系统镜像，分别命名为Image、copy1和copy2，这三份镜像文件将通过flash脚本烧录到AGX设备的EMMC存储单元中。

（2）在Cboot阶段的执行逻辑中，文件系统可通过文件路径确定文件的存储位置，分别找到三份镜像文件在EMMC的存储位置，对三份镜像文件进行CRC校验，通过比较三份镜像文件的CRC校验码，可判断镜像文件是否发生比特翻转错误。

（3）如果三份镜像文件的CRC校验码相同，说明镜像文件并未发生翻转错误，可选择原始镜像文件（Image）加载到内存中。

（4）如果只有两份镜像文件的CRC校验码相同，说明有一份镜像文件发生比特翻转错误，可从CRC校验码相同的两份镜像文件中选择一份加载到内存中。

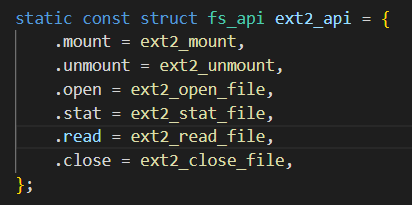
（5）如果三份镜像文件的CRC校验码均不相同，说明至少有两份镜像文件发生翻转错误，无法判断哪份镜像文件是正确的，此时对三份镜像文件逐个按位比较，从三份镜像文件中逐个选取正确的每一位组成一份正确的镜像文件，将组成的正确镜像文件加载到内存中，具体实现思路如图所示：

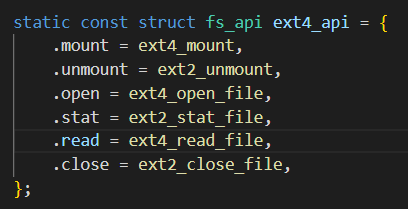


由于堆栈空间有限，无法分配三块内存空间将Image、copy1、copy2分别复制到内存中（内存中负责加载镜像文件的大小为64MB，一份镜像文件的大小为34MB左右，实际上无法将两份镜像文件复制加载到内存中），所以设计分配两块缓冲区temp\_buf1和temp\_buf2（目前为temp\_buf1和temp\_buf2分配的大小为2000000字节，尚不清楚temp\_buf1和temp\_buf2大小的上限），先将copy1文件复制加载到内存中，再将copy1文件的一块（2000000字节大小）复制到temp\_buf1，同理，先将copy2文件复制加载到内存中，再将copy2文件的一块（2000000字节大小）复制到temp\_buf1，最后将Image文件复制到内存中，然后逐个字节比较Image文件、temp\_buf1、temp\_buf2，将Image镜像文件发生错误的字节位置和正确字节记录下来（目前记录错误字节位置数组和正确字节数组的大小为1000000，也就是说目前默认发生翻转的错误字节不会超过1000000个，后续可以根据需求优化），重复上述过程（大概需要循环17次，所以此种错误处理需要时间，大约10s左右），等三个镜像文件全部比较完，根据记录下来的错误字节位置将正确字节写入内存中的对应位置，最后输出内存中组成的正确镜像文件的CRC校验码。

（6）将正确的镜像文件复制到内存中，解压缩提取镜像文件后，就可以成功启动操作系统内核。

（7）**由于Cboot内部未提供文件系统的创建文件和写文件API**，如图所示，文件系统（ext2或ext4）只提供打开文件和读文件等API，创建文件或写文件的API并未提供，所以无法在Cboot阶段覆盖重写发生翻转错误的镜像文件，将错误镜像文件的覆盖重写操作放到操作系统启动后执行。





（8）成功启动操作系统内核后，需要定期对boot文件夹下的三份镜像文件进行CRC校验，通过比较三份镜像文件的CRC校验码，可判断镜像文件是否发生比特翻转错误，如果发现存在镜像文件发生比特翻转错误，能够自动重写覆盖错误镜像文件。

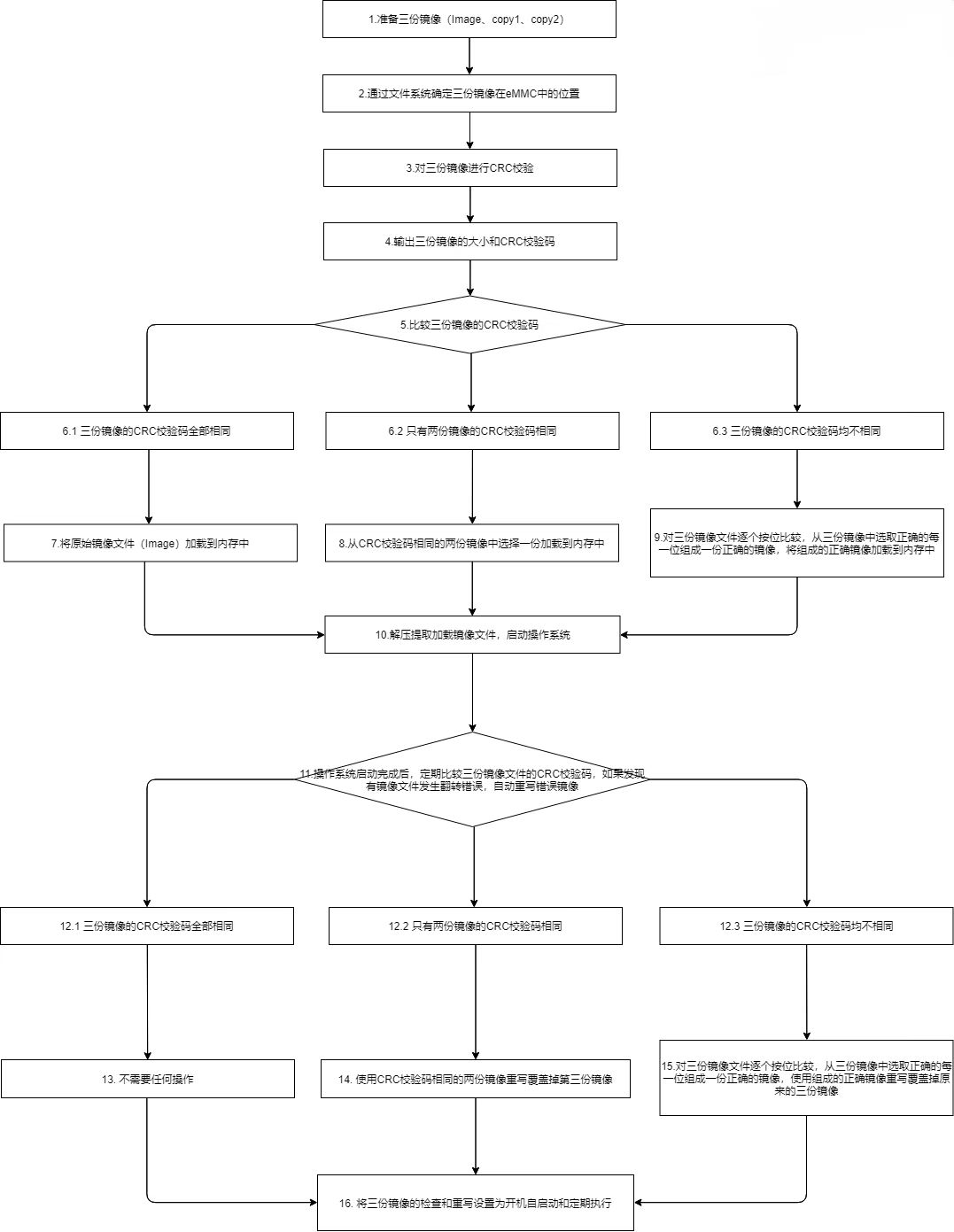
（9）如果三份镜像文件的CRC校验码相同，不需要进行任何容错操作。

（10）如果只有两份镜像文件的CRC校验码相同，需要使用两份正确的镜像文件覆盖重写掉第三份错误镜像文件。

（11）如果三份镜像文件的CRC校验码均不相同，说明至少有两份镜像文件发生比特翻转错误，无法判断哪份镜像文件是正确的，此时对三份镜像文件逐个按位比较，从三份镜像中逐个选取正确的每一位组成一份正确的镜像文件，使用组成的正确镜像文件重写覆盖掉原来的三份镜像文件。

（12）将三份镜像文件的检查校验和覆盖重写脚本设置为开机自启动和定期自动执行。

Cboot实现Kernel镜像加固的设计思路见下图：



1. **Cboot加固方案测试**
   1. **脚本功能**

在AGX设备的/boot文件夹下准备四个脚本，分别是Check.c、Test1.c、Test2.c、Test3.c。想要使用它们时，先进行gcc编译，然后执行，例如：

$ gcc Check.c -o Check

$ ./Check

* + 1. **主脚本功能**

Check.c是检查重写脚本，它的作用是检查对比Image、copy1和copy2文件，如果发现三个镜像文件的CRC校验码不相同，就进行不同的错误处理。如果只有两个镜像文件的CRC校验码相同，就使用CRC校验码相同的镜像文件覆盖重写掉第三个镜像文件，如果三个镜像文件的CRC校验码均不相同，对三份镜像文件逐个按位比较，从三份镜像中逐个选取正确的每一位组成一份正确的镜像文件，使用组成的正确镜像文件重写覆盖掉原来的三份镜像文件。覆盖重写错误镜像文件完成后，将检查信息写入到result.txt文件中，result.txt文件默认放在/boot文件夹下。

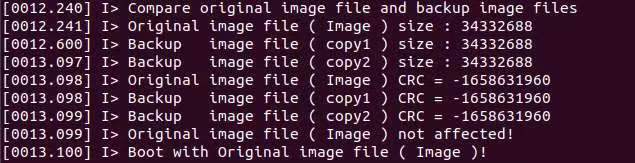
* + 1. **辅助脚本功能**

Test1.c的作用是将Image镜像文件的10到20位字节翻转，Test2.c的作用是将copy1镜像文件的20到30位字节翻转，Test3.c的作用是将copy2镜像文件的30到40位字节翻转，这三个文件用来测试将镜像文件的某些比特位翻转后，Cboot能否正常容错重启。

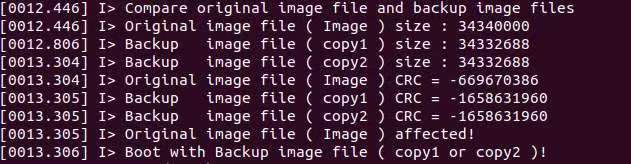
**2.2效果展示**

**2.2.1 Cboot加固效果展示**

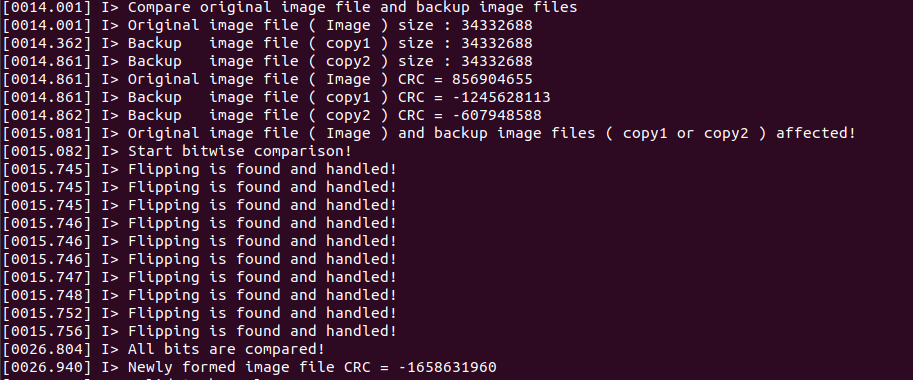
在Cboot层面，当三个镜像全部一致时：



当只有两个镜像保持一致时：



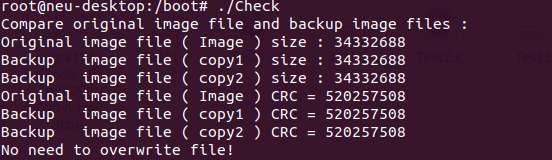
当三个镜像全部不一致时：



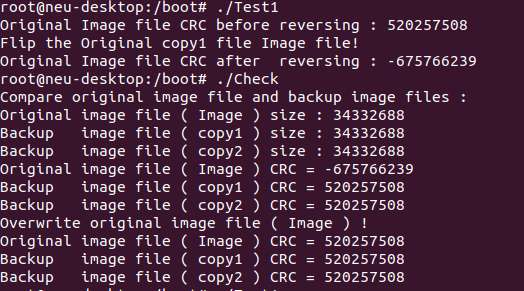
当三个镜像全部不一致时，需要三个镜像逐个字节进行比对，Image镜像有34332688个字节，所以需要一点时间，大概10s左右。

**2.2.2脚本效果展示**

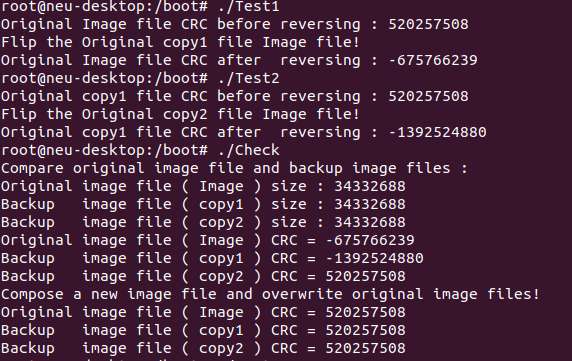
在Linux层面，当三个镜像全部一致时：



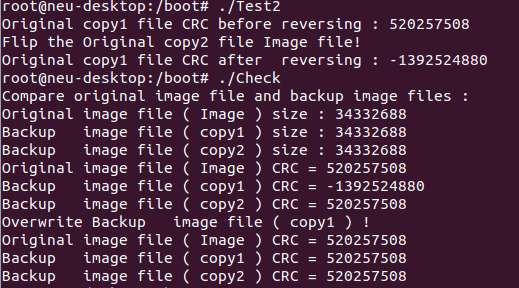
翻转Image镜像并调用检查重写脚本：



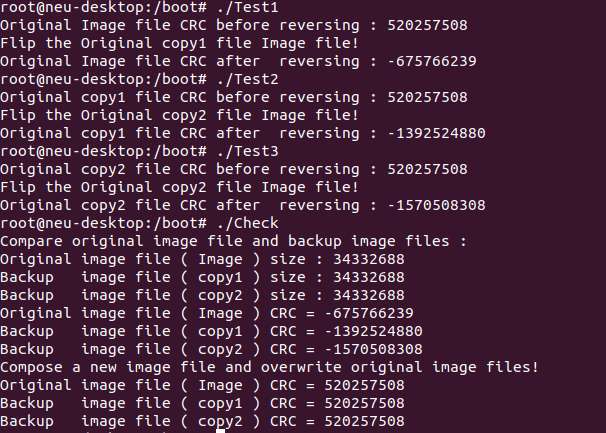
翻转Image镜像和copy1镜像并调用检查重写脚本：



翻转copy1镜像并调用检查重写脚本：



翻转Image镜像、copy1镜像和copy2镜像并调用检查重写脚本：



经过上面五轮调用检查重写脚本，得到的信息写入到result.txt文件：

