# 实验8鸿蒙LiteOS-a根文件系统制作

1. 实验目的
2. 实验内容

# 1.根文件系统内容与制作

## 1.1 最终结果

本章节做的修改会制作为补丁文件：

* 05\_openharmony\_rootfs.patch

假设目录openharmony中是未修改的代码，从没打过补丁； 假设补丁文件放在openharmony的同级目录；  
 对于STM32MP157，打补丁方法如下

：

$ cd openharmony  
$ patch -p1 < ../openharmony\_100ask\_v1.2.patch  
$ patch -p1 < ../01\_openharmony\_add\_demo\_board.patch  
$ patch -p1 < ../02\_openharmony\_memmap\_stm32mp157.patch   
$ patch -p1 < ../03\_openharmony\_uart\_stm32mp157.patch  
$ patch -p1 < ../04\_openharmony\_ramfs\_stm32mp157.patch  
$ patch -p1 < ../05\_openharmony\_rootfs.patch

* 1
* 2
* 3
* 4
* 5
* 6
* 7

对于IMX6ULL，打补丁方法如下：

$ cd openharmony  
$ patch -p1 < ../openharmony\_100ask\_v1.2.patch  
$ patch -p1 < ../01\_openharmony\_add\_demo\_board.patch  
$ patch -p1 < ../02\_openharmony\_memmap\_imx6ull.patch   
$ patch -p1 < ../03\_openharmony\_uart\_imx6ull.patch  
$ patch -p1 < ../04\_openharmony\_ramfs\_imx6ull.patch  
$ patch -p1 < ../05\_openharmony\_rootfs.patch

* 1
* 2
* 3
* 4
* 5
* 6
* 7

打上补丁后，可以如此编译(对于STM32MP157、IMX6ULL，编译命令是一样的)：

$ cd kernel/liteos\_a  
$ cp tools/build/config/debug/demochip\_clang.config .config  
$ make clean  
$ make  
$ make rootfs

* 1
* 2
* 3
* 4
* 5

## 1.2 根文件系统的内容

看看一个简单的程序：

#include <stdio.h>  
int main(int argc, char \*\*argv)  
{  
        printf("hello, world!\n");  
 return 0;  
}

* 1
* 2
* 3
* 4
* 5
* 6

可以编译出一个APP：hello。  
 有几个问题要考虑：

* printf不是我们实现的，它在哪？
* hello放在板子上后，怎么启动它？能否自动启动？

解决这几个问题后，就可以知道根文件系统的内容了：

* /lib：库，比如printf函数就是在库里的
* /bin：APP，hello这样的程序放在/bin或/usr/bin这些目录里
* 至少有这些APP：
* init：内核启动的第一个APP，它会去启动其他APP，比如shell
* shell：也是一个APP，可以让我们输入各类命令
* 我们自己的APP：比如hello
* /etc：想自动启动APP怎么办？应该有配置文件，init进程根据配置文件去启动其他APP
* 比如/etc/init.cfg
* /dev：设备节点，在Liteos-a中不需要我们自己创建

## 1.3 根文件系统的制作

### 1.3.1 Makefile分析

在kernnel/liteos\_a目录执行make help：

book@100ask:~/openharmony/kernel/liteos\_a$ make help  
-------------------------------------------------------  
1.====make help: get help infomation of make  
2.====make: make a debug version based the .config  
3.====make debug: make a debug version based the .config  
4.====make release: make a release version for all platform  
5.====make release PLATFORM=xxx: make a release version only for platform xxx  
6.====make rootfsdir: make a original rootfs dir  
7.====make rootfs FSTYPE=\*\*\*: make a original rootfs img  
8.====make test: make the testsuits\_app and put it into the rootfs dir  
9.====make test\_apps FSTYPE=\*\*\*: make a rootfs img with the testsuits\_app in it  
xxx should be one of (hi3516cv300 hi3516ev200 hi3556av100/cortex-a53\_aarch32 hi3559av100/cortex-a53\_aarch64)  
\*\*\* should be one of (jffs2)  
-------------------------------------------------------

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14

可以知道：执行make rootfs可以制作根文件系统。 分析Makefile确定它的制作过程。

#### *1. ROOTFS目标：*

$(ROOTFS): $(ROOTFSDIR)  
        $(HIDE)$(shell $(LITEOSTOPDIR)/tools/scripts/make\_rootfs/rootfsimg.sh $(ROOTFS\_DIR) $(FSTYPE) ${ROOTFS\_SIZE})  
        $(HIDE)cd $(ROOTFS\_DIR)/.. && zip -r $(ROOTFS\_ZIP) $(ROOTFS)  
ifneq ($(OUT), $(LITEOS\_TARGET\_DIR))  
        $(HIDE)mv $(ROOTFS\_DIR) $(LITEOS\_TARGET\_DIR)rootfs  
endif  
  
$(ROOTFSDIR): prepare $(APPS)  
        $(HIDE)$(MAKE) clean -C apps  
        $(HIDE)$(shell $(LITEOSTOPDIR)/tools/scripts/make\_rootfs/rootfsdir.sh $(OUT)/bin $(OUT)/musl $(ROOTFS\_DIR))  
ifneq ($(VERSION),)  
        $(HIDE)$(shell $(LITEOSTOPDIR)/tools/scripts/make\_rootfs/releaseinfo.sh "$(VERSION)" $(ROOTFS\_DIR))  
endif  
  
prepare:  
        $(HIDE)mkdir -p $(OUT)/musl  
ifeq ($(LOSCFG\_COMPILER\_CLANG\_LLVM), y)  
        $(HIDE)cp -f $(LITEOSTOPDIR)/../../prebuilts/lite/sysroot/usr/lib/$(LLVM\_TARGET)/a7\_softfp\_neon-vfpv4/libc.so $(OUT)/musl  
        $(HIDE)cp -f $(LITEOS\_COMPILER\_PATH)/lib/$(LLVM\_TARGET)/c++/a7\_softfp\_neon-vfpv4/libc++.so $(OUT)/musl  
else  
        $(HIDE)cp -f $(LITEOS\_COMPILER\_PATH)/target/usr/lib/libc.so $(OUT)/musl  
        $(HIDE)cp -f $(LITEOS\_COMPILER\_PATH)/arm-linux-musleabi/lib/libstdc++.so.6 $(OUT)/musl  
        $(HIDE)cp -f $(LITEOS\_COMPILER\_PATH)/arm-linux-musleabi/lib/libgcc\_s.so.1 $(OUT)/musl  
        $(STRIP) $(OUT)/musl/\*  
endif  
  
$(APPS): $(LITEOS\_TARGET)  
        $(HIDE)$(MAKE) -C apps all  
        



* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30

* ROOTFSDIR：使用rootfsdir.sh创建一些目录
* prepare：复制libc.so、libc++.so
* APPS：进入apps目录执行make all

#### *2. 编译APP*

有目录：kernel/liteos\_a/apps，这个目录下有：

* module.mk：定义了

APP\_SUBDIRS += shell  
APP\_SUBDIRS += init

* 1
* 2

* Makefile：

$(APPS):  
ifneq ($(APP\_SUBDIRS), )  
 $(HIDE) for dir in $(APP\_SUBDIRS); do $(MAKE) -C $$dir ; done  
endif

* 1
* 2
* 3
* 4

就是再次进入shell、init目录，执行make命令，去变量shell程序、init程序。

### 1.3.2 演示

# 2.正式版本的init进程

Liteos-a中有两个init程序：

* 测试版本：kernel\liteos\_a\apps\init\src\init.c
* 正式版本：base\startup\services\init\_lite\src\main.c

## 2.1 测试版本

源码：kernel\liteos\_a\apps\init\src\init.c  
 我们在kernel\liteos\_a目录下执行make rootfs时使用的就是测试版本，  
 它的功能很简单：只是启动/bin/shell程序，源码如下：

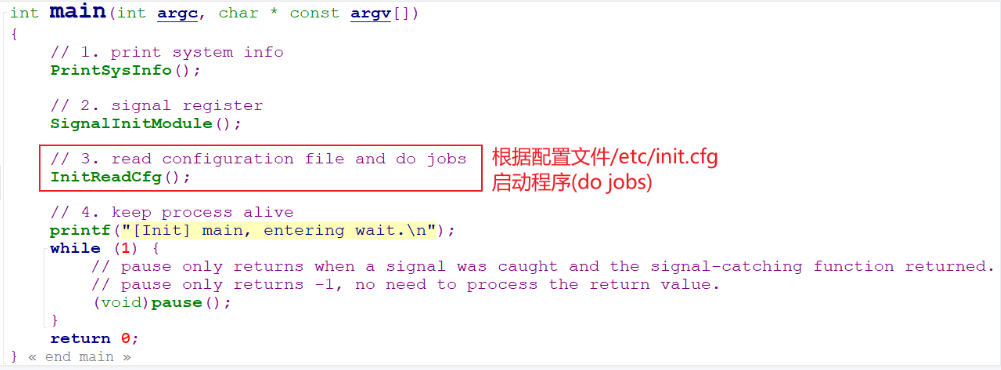
int main(int argc, char \* const \*argv)  
{  
 int ret;  
 const char \*shellPath = "/bin/shell";  
  
 ret = fork();  
 if (ret < 0) {  
 printf("Failed to fork for shell\n");  
 } else if (ret == 0) {  
 (void)execve(shellPath, NULL, NULL);  
 exit(0);  
 }  
  
 while (1) {  
 ret = waitpid(-1, 0, WNOHANG);  
 if (ret == 0) {  
 sleep(1);  
 }  
 };  
}



* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20

## 2.2 正式版本

源码：base\startup\services\init\_lite\src\main.c



怎么单独编译正式版本，尚未研究。  
 可以使用这样的命令去编译：python build.py ipcamera\_hi3518ev300 -b debug 可以得到rootfs目录，里面有/bin/init, /etc/init.cfg等文件。

### 2.2.1 配置文件

#### *1. 分析配置文件*

配置文件中内容分为两部分：

* services：定义了多个服务，它对应某些APP
* jobs：可以定义一些APP，也可去启动服务
* pre-init：预先执行的初始化
* init：初始化
* post-init：最后的初始化

#### *2. 示例*

./vendor/huawei/camera/init\_configs/init\_liteos\_a\_3516dv300.cfg  
./vendor/huawei/camera/init\_configs/init\_liteos\_a\_3518ev300.cfg