实验 8 鸿蒙 LiteOS-a 根文件系统制作

22920212204396　　黄子安

一、实验目的

1、 为 demochip 单板移植根文件系统

2、 通过实验的方式进一步理解操作系统的文件系统

二、实验环境

1、物理机：Windows

2、虚拟机：Ubuntu 18.04.6

3、开发板：imx6ull mini

三、实验内容

制作根文件系统的两个步骤：先创建文件夹 rootfs，在其中填充内容，例如/lib，/bin，即程序的 bin 文件和相关的库文件；之后将这个文件制作成镜像文件烧写到开发板中

执行 make rootfs 的过程：执行 make rootfs 的时候会先去执行目标 rootfsdir

```
$(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 依赖于 prepare 和 apps，其中 prepare 完成一些准备工作，先创建对应的目录，之后拷贝库文件，apps 又依赖于 liteos_target，apps 的作用是进入 apps 目录，之后执行 all 目标

```
$(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
    $(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
```

All 目标里面会执行对应文件夹里的全部 make，也就是对 shell 和 init 进行编译，生成对应的应用程序

```
all: $(APPS)

# Make
$(APPS):
ifneq ($(APP_SUBDIRS), )
    $(HIDE) for dir in $(APP_SUBDIRS); do $(MAKE) -C $$dir ; done
endif
```

```
APP_SUBDIRS :=

##compile modules config##

ifeq ($(LOSCFG_SHELL), y)
APP_SUBDIRS += shell
endif

ifeq ($(LOSCFG_USER_INIT_DEBUG), y)
APP_SUBDIRS += init
endif
```

之后执行对应的脚本制作镜像文件，脚本文件如下所示，会生成 shell 和 init 对应的 jffs2 文件

```
$(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
```

```
system=$(uname -s)
ROOTFS_DIR=$1
FSTYPE=$2
ROOTFS_SIZE=$3
ROOTFS_IMG=${ROOTFS_DIR}".img"
JFFS2_TOOL=$(dirname $(readlink -f "$0"))/../../fsimage/mkfs.jffs2
WIN_JFFS2_TOOL=$(dirname $(readlink -f "$0"))/../../fsimage/win-x86/mkfs.jffs2.exe

if [ "${ROOTFS_DIR}" = "*rootfs" ]; then
    chmod -R 755 ${ROOTFS_DIR}
    chmod 700 ${ROOTFS_DIR}/bin/init 2> /dev/null
    chmod 700 ${ROOTFS_DIR}/bin/shell 2> /dev/null
fi

if [ "${FSTYPE}" = "jffs2" ]; then
    if [ "${system}" != "Linux" ] ; then
        ${WIN_JFFS2_TOOL} -q -o ${ROOTFS_IMG} -d ${ROOTFS_DIR} --pagesize=4096
    else
        chmod +x ${JFFS2_TOOL}
        echo ${JFFS2_TOOL} -q -o ${ROOTFS_IMG} -d ${ROOTFS_DIR} --pagesize=4096 --pad=${ROOTFS_SIZE}
        ${JFFS2_TOOL} -q -o ${ROOTFS_IMG} -d ${ROOTFS_DIR} --pagesize=4096 --pad=${ROOTFS_SIZE}
    cp ${ROOTFS_IMG} ${ROOTFS_DIR}".jffs2"
    cp ${ROOTFS_IMG} ${ROOTFS_DIR}".jffs2.bin"
    fi
```

之后存在一个问题，导出的 rootfs 的 jffs2 文件中有 2M，但是模拟的外存有 10M，后面的 8M 会被变为随机值，运行的时候会警告，解决这个问题有两种方法：1、在 uboot 烧写前将对应的 10M 都设置为 ff，2、使用 uboot 将其扩充成 10M（后面 8M 全写 ff）；imx6ull 使用了第一种，157 单板使用了第二种

直接 make rootfs，可以发现生成的 rootfs 大小是 10M，但是理论上应该是 2M

```
-rw-r--r--  1 book book 10485760 Dec  6 13:21 rootfs.img
-rw-r--r--  1 book book 10485760 Dec  6 13:21 rootfs.jffs2
-rw-r--r--  1 book book 10485760 Dec  6 13:21 rootfs.jffs2.bin
-rw-rw-r--  1 book book   863640 Dec  6 13:21 rootfs.zip
```

原因在于在一开始的 makefile 中指定了 rootfs 的大小是 10M

```
ifeq ($(LOSCFG_PLATFORM_STM32MP157), y)
FSTYPE = jffs2
ROOTFS_SIZE = 0xA00000
endif
ifeq ($(LOSCFG_PLATFORM_DEMOCHIP), y)
```

在脚本打成镜像的时候 rootfs 会被扩充成 10M，因此重新定义个语句生成 2M 的 rootfs

```
if [ "${FSTYPE}" = "jffs2" ]; then
    if [ "${system}" != "Linux" ] ; then
        ${WIN_JFFS2_TOOL} -q -o ${ROOTFS_IMG} -d ${ROOTFS_DIR} --pagesize=4096
    else
        chmod +x ${JFFS2_TOOL}
        echo ${JFFS2_TOOL} -q -o ${ROOTFS_IMG} -d ${ROOTFS_DIR} --pagesize=4096 --pad=${ROOTFS_SIZE}
        ${JFFS2_TOOL} -q -o ${ROOTFS_IMG} -d ${ROOTFS_DIR} --pagesize=4096 --pad=${ROOTFS_SIZE}
    cp ${ROOTFS_IMG} ${ROOTFS_DIR}".jffs2"
    cp ${ROOTFS_IMG} ${ROOTFS_DIR}".jffs2.bin"
    fi
```

定义个新的变量 ROOTFS_JFFS2，后缀为.jffs2

```
ROOTFS_SIZE=$3
ROOTFS_IMG=${ROOTFS_DIR}".img"
ROOTFS_JFFS2=${ROOTFS_DIR}".jffs2"
JFFS2_TOOL=$(dirname $(readlink -f "$0"))/../../fsimage/mkfs.jffs2
WIN_JFFS2_TOOL=$(dirname $(readlink -f "$0"))/../../fsimage/win-x86/mkfs.jffs2.exe
```

之后打出一个新的镜像文件，不进行大小的扩充

```
if [ "${FSTYPE}" = "jffs2" ]; then
    if [ "${system}" != "Linux" ] ; then
        ${WIN_JFFS2_TOOL} -q -o ${ROOTFS_IMG} -d ${ROOTFS_DIR} --pagesize=4096
    else
        chmod +x ${JFFS2_TOOL}
        echo ${JFFS2_TOOL} -q -o ${ROOTFS_IMG} -d ${ROOTFS_DIR} --pagesize=4096 --pad=${ROOTFS_SIZE}
        ${JFFS2_TOOL} -q -o ${ROOTFS_IMG} -d ${ROOTFS_DIR} --pagesize=4096 --pad=${ROOTFS_SIZE}
        ${JFFS2_TOOL} -q -o ${ROOTFS_JFFS2} -d ${ROOTFS_DIR} --pagesize=4096
    cp ${ROOTFS_IMG} ${ROOTFS_DIR}".jffs2"
    cp ${ROOTFS_IMG} ${ROOTFS_DIR}".jffs2.bin"
```

因为已经直接生成了 jffs2，因此这里无需要在使用原来的 cp 命令来复制生成 jffs2 文件

```
    if [ "${system}" != "Linux" ] ; then
        ${WIN_JFFS2_TOOL} -q -o ${ROOTFS_IMG} -d ${ROOTFS_DIR} --pagesize=4096
    else
        chmod +x ${JFFS2_TOOL}
        echo ${JFFS2_TOOL} -q -o ${ROOTFS_IMG} -d ${ROOTFS_DIR} --pagesize=4096 --pad=${ROOTFS_SIZE}
        ${JFFS2_TOOL} -q -o ${ROOTFS_IMG} -d ${ROOTFS_DIR} --pagesize=4096 --pad=${ROOTFS_SIZE}
        ${JFFS2_TOOL} -q -o ${ROOTFS_JFFS2} -d ${ROOTFS_DIR} --pagesize=4096
    cp ${ROOTFS_IMG} ${ROOTFS_DIR}".jffs2.bin"
    fi
```

再次执行 make rootfs 可以看到生成的 rootfs 小了很多

```
-rw-r--r--  1 book book 10485760 Dec  6 13:53 rootfs.img
-rw-r--r--  1 book book  1043204 Dec  6 13:53 rootfs.jffs2
-rw-r--r--  1 book book 10485760 Dec  6 13:53 rootfs.jffs2.bin
-rw-rw-r--  1 book book   863640 Dec  6 13:53 rootfs.zip
```

init 进程的作用：读取/etc 目录下的配置文件，之后用于启动对应的应用程序，测试版本的 init 进程只会启动 shell 程序，不会去根据配置文件启动别的程序，因此正式版本的需要在这个基础上扩充

```
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);
        }
    };
}
```

正式版本的 init 程序如下所示，会先去读取配置文件，之后执行对应的任务；配置文件中内容可以分为两个部分：services 中定义了多个服务，对应某些 APP，jobs 可以去定义一些 APP，也可以去启动服务，包括 pre-init 预先执行的初始化，init 初始化操作和 post-init 即最后的初始化操作

init 进程最后启动了配置文件中被定义好的别的程序，从而整个操作系统完成启动工作。

```
int main(int argc, char * const argv[])
{
    // 1. print system info
    PrintSysInfo();

    // 2. signal register
    SignalInitModule();

    // 3. read configuration file and do jobs
    InitReadCfg();

    // 4. keep process alive
    printf("[Init] main, entering wait.\n");
    while (1) {
        // pause only returns when a signal was caught and the signal-catching function returned.
        // pause only returns -1, no need to process the return value.
        (void)pause();
    }
    return 0;
}
```

四、实验心得

本次实验对 Litsos-a 的根文件进行了介绍，阅读了将对应文件夹制作成 rootfs.jffs2 的 makefile 执行过程，从而对 Liteos 一开机便进入 shell 程序的过程有初步了解，此外也介绍了 init 进程启动其他进程的步骤，通过配置文件来对程序启动进行控制，从操作系统到进程运行跨进了一步，加深了对于 Liteos 的理解。