实用操作系统课程实验报告

实验名称：	实验八 鸿蒙 LiteOS-a 内核移植——根文件系统制作
实验日期：	2023-12-22
实验地点：	文宣楼 B313

学号：	33920212204567
姓名：	任宇
专业年级：	软工 2021 级
学年学期：	2023-2024 学年第一学期

1.实验目的

- 进行鸿蒙 LiteOS-a 内核的根文件系统制作

2. 实验内容和步骤

（1） 查看 LiteOS-a 内核是如何进行根文件系统的制作：

输入 make help 指令，查看 make 指令如何构建根文件系统，如图所示：

```
book@ry-virtual-machine:~/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)
```


查看 Makefile 文件：

```
180 $(APPS): $(LITEOS_TARGET)
181   $(HIDE)$(MAKE) -C apps all
182
183 prepare:
184   $(HIDE)mkdir -p $(OUT)/musl
185 ifeq ($(LOSCFG_COMPILER_CLANG_LLVM), y)
186   $(HIDE)cp -f $(LITEOSTOPDIR)/../../prebuilts/lite/sysroot/usr/lib/$(LLVM_TARGET)/a7_softfp_neon-vfpv4/libc.so $(OUT)/musl
187   $(HIDE)cp -f $(LITEOS_COMPILER_PATH)/lib/$(LLVM_TARGET)/c++/a7_softfp_neon-vfpv4/libc++.so $(OUT)/musl
188 else
189   $(HIDE)cp -f $(LITEOS_COMPILER_PATH)/target/usr/lib/libc.so $(OUT)/musl
190   $(HIDE)cp -f $(LITEOS_COMPILER_PATH)/arm-linux-musleabi/lib/libstdc++.so.6 $(OUT)/musl
191   $(HIDE)cp -f $(LITEOS_COMPILER_PATH)/arm-linux-musleabi/lib/libgcc_s.so.1 $(OUT)/musl
192   $(STRIP) $(OUT)/musl/*
193 endif
194
195 $(ROOTFSDIR): prepare $(APPS)
196   $(HIDE)$(MAKE) clean -C apps
197   $(HIDE)$(shell $(LITEOSTOPDIR)/tools/scripts/make_rootfs/rootfsdir.sh $(OUT)/bin $(OUT)/musl $(ROOTFS_DIR))
198 ifneq ($(VERSION),)
199   $(HIDE)$(shell $(LITEOSTOPDIR)/tools/scripts/make_rootfs/releaseinfo.sh "$(VERSION)" $(ROOTFS_DIR))
200 endif
201
202 $(ROOTFS): $(ROOTFSDIR)
203   $(HIDE)$(shell $(LITEOSTOPDIR)/tools/scripts/make_rootfs/rootfsimg.sh $(ROOTFS_DIR) $(FSTYPE)  ${ROOTFS_SIZE})
204   $(HIDE)cd $(ROOTFS_DIR)/.. && zip -r $(ROOTFS_ZIP) $(ROOTFS)
205 ifneq ($(OUT), $(LITEOS_TARGET_DIR))
206   $(HIDE)mv $(ROOTFS_DIR) $(LITEOS_TARGET_DIR)rootfs
207 endif
```

可见其 prepare 的过程为：首先创建目录，接着拷贝库文件。而 prepare 依赖于 APPS，APPS 会进入 apps 目录执行 make all。在 apps 目录下有 module.mk 文件，其中定义了

- APP_SUBDIRS += shell
- APP_SUBDIRS += init

在 make 过程中会进入 shell 和 init 目录，执行 make 命令并

编译 shell 程序和 init 程序。

（2） 了解过程后，修改 Makefile 文件：

将文件系统类型由 vfat 改为 jffs2：

```
ifeq ($(LOSCFG_PLATFORM_DEMOCHIP), y)
FSTYPE = jffs2
#####   FSTYPE = vfat   #####
ROOTFS_SIZE = 0xA00000
```

（3） 修改 rootfsimg.sh 脚本：

添加一个新的变量 ROOTFS_JFFS2，用来表示 JFFS2 文件系统的路径。接着修改 JFFS2 文件系统创建部分的代码：原先的脚本在创建 JFFS2 文件系统后，会将生成的文件系统镜像复制为 .jffs2 后缀的文件。而修改后的脚本使用 JFFS2 工具直接生成一个 .jffs2 后缀的文件系统镜像，而不再是复制生成。

```
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
```

```
        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"
```

（4） 制作 rootfs，验证修改结果：

```
book@ry-virtual-machine:~/openharmony_demo/kernel$ cd liteos_a
book@ry-virtual-machine:~/openharmony_demo/kernel/liteos_a$ cp tools/build/config/debug/demochip_clang.config .config
book@ry-virtual-machine:~/openharmony_demo/kernel/liteos_a$ make rootfs
make[1]: 进入目录“/home/book/openharmony_demo/kernel/liteos_a/arch/arm/arm”
make[1]: 对“all”无需做任何事。
make[1]: 离开目录“/home/book/openharmony_demo/kernel/liteos_a/arch/arm/arm”
make[1]: 进入目录“/home/book/openharmony_demo/kernel/liteos_a/platform”
make[1]: 离开目录“/home/book/openharmony_demo/kernel/liteos_a/platform”
make[1]: 进入目录“/home/book/openharmony_demo/kernel/liteos_a/kernel/common”
make[1]: 对“all”无需做任何事。
make[1]: 离开目录“/home/book/openharmony_demo/kernel/liteos_a/kernel/common”
  adding: rootfs/bin/ (stored 0%)
  adding: rootfs/bin/shell (deflated 60%)
  adding: rootfs/bin/init (deflated 88%)
  adding: rootfs/etc/ (stored 0%)
  adding: rootfs/lib/ (stored 0%)
  adding: rootfs/lib/libc++.so (deflated 71%)
  adding: rootfs/lib/libc.so (deflated 45%)
book@ry-virtual-machine:~/openharmony_demo/kernel/liteos_a$
```

到 out/demochip/目录下查看制作的 rootfs 文件，可以观察到

rootfs.jffs2.bin 文件被成功扩充：

```
book@ry-virtual-machine:~/openharmony_demo/kernel/liteos_a$ cd out/demochip/
book@ry-virtual-machine:~/openharmony_demo/kernel/liteos_a/out/demochip$ ls -l
总用量 33692
drwxrwxr-x  2 book book     4096 11月 29 23:49 bin
drwxrwxr-x  2 book book     4096 11月 29 23:49 lib
-rwxrwxr-x  1 book book  1069216 11月 29 23:49 liteos
-rw-rw-r--  1 book book  8821442 11月 29 23:49 liteos.asm
-rwxrwxr-x  1 book book   891776 11月 29 23:49 liteos.bin
-rw-rw-r--  1 book book   540715 11月 29 23:49 liteos.map
-rw-rw-r--  1 book book   265998 11月 29 23:49 liteos.sym.sorted
drwxrwxr-x  2 book book     4096 11月 29 23:49 musl
drwxrwxr-x 16 book book     4096 11月 29 21:47 obj
drwxrwxr-x  9 book book     4096 11月 29 23:49 rootfs
-rw-r--r--  1 book book 10485760 11月 29 23:49 rootfs.img
-rw-r--r--  1 book book  1043204 11月 29 23:49 rootfs.jffs2
-rw-r--r--  1 book book 10485760 11月 29 23:49 rootfs.jffs2.bin
-rw-rw-r--  1 book book   863634 11月 29 23:49 rootfs.zip
book@ry-virtual-machine:~/openharmony_demo/kernel/liteos_a/out/demochip$
```

（5） LiteOS-a 中的 init 程序：

在 LiteOS-a 的内核中一共有两个 init 程序，分别为测试版本的 init 程序和正式版本的 init 程序。

对于测试版本的 init 程序，其源码位于 kernel\liteos_a\apps\init\src\init.c 文件，它只负责启动 shell 程序，并不读取配置文件：

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

对于正式版本的 init 程序，其源码位于 base\startup\services\init_lite\src\main.c 文件中，其会根据配置文件 init.cfg 启动程序：

```c
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;
}
```

InitReadCfg 函数会读取配置文件，并进行初始化，具体如图所示：

```c
void InitReadCfg()
{
    // read configuration file in json format
    char* fileBuf = ReadFileToBuf();
    if (fileBuf == NULL) {
        printf("[Init] InitReadCfg, read file %s failed! err %d.\n", INIT_CONFIGURATION_FILE, errno);
        return;
    }

    cJSON* fileRoot = cJSON_Parse(fileBuf);
    free(fileBuf);
    fileBuf = NULL;

    if (fileRoot == NULL) {
        printf("[Init] InitReadCfg, parse failed! please check file %s format.\n", INIT_CONFIGURATION_FILE);
        return;
    }

    // parse services
    ParseAllServices(fileRoot);

    // parse jobs
    ParseAllJobs(fileRoot);

    // release memory
    cJSON_Delete(fileRoot);

    // do jobs
    DoJob("pre-init");
    DoJob("init");
    DoJob("post-init");
    ReleaseAllJobs();
}
```

配置文件内容可以分为两大部分：服务（services）和工作（jobs）

Jobs又可以分为3类，pre-init（预先执行的初始化）、init(初始化)以及post-init（后初始化）

分析配置文件，以/vendor/huawei/camera/init_configs/init_liteos_a_3516dv300.cfg 为例：

可以发现，Jobs 部分主要包含一些命令，而 Services 部分则是对服务的定义，例如 shell 程序：

```
{
    "jobs" : [{
            "name" : "pre-init",
            "cmds" : [
                "mkdir /storage/data/log",
                "chmod 0755 /storage/data/log",
                "chown 4 4 /storage/data/log",
                "mkdir /storage/data/softbus",
                "chmod 0700 /storage/data/softbus",
                "chown 7 7 /storage/data/softbus",
                "mkdir /sdcard",
                "chmod 0777 /sdcard",
                "mount vfat /dev/mmcblk0 /sdcard rw,umask=000",
                "mount vfat /dev/mmcblk1 /sdcard rw,umask=000"
            ]
        }, {
            "name" : "init",
            "cmds" : [
                "start shell",
                "start apphilogcat",
                "start foundation",
                "start bundle_daemon",
                "start appspawn",
                "start media_server",
                "start wms_server"
            ]
        }, {
            "name" : "post-init",
            "cmds" : [
                "chown 0 99 /dev/dev_mgr",
                "chown 0 99 /dev/hdfwifi",
                "chown 0 99 /dev/gpio",
                "chown 0 99 /dev/i2c-0",
                "chown 0 99 /dev/i2c-1",
                "chown 0 99 /dev/i2c-2",
                "chown 0 99 /dev/i2c-3",
```

启动shell程序

```
52      "services" : [{
53              "name" : "foundation",
54              "path" : "/bin/foundation",
55              "uid" : 7,
56              "gid" : 7,
57              "once" : 0,
58              "importance" : 1,
59              "caps" : [10, 11, 12, 13]
60          }, {
61              "name" : "shell",
62              "path" : "/bin/shell",
63              "uid" : 2,
64              "gid" : 2,
65              "once" : 0,
66              "importance" : 0,
67              "caps" : [4294967295]
68          }, {
69              "name" : "appspawn",
70              "path" : "/bin/appspawn",
```

对shell的定义

3. 实验总结

本次实验的主要目的是进行鸿蒙 LiteOS-a 内核的根文件系统制作。实验过程中，我首先了解了 LiteOS-a 内核构建根文件系统的方式，接着对相关的 Makefile 和脚本进行了修改，最后制作了 rootfs 并验证了修改结果。通过此次实验，我不仅掌握了 LiteOS-a 内核根文件系统的制作流程，还学习了如何通过修改 Makefile 和脚本来适应不同的文件系统需求。这对于理解操作系统的启动过程和文件系统的配置具有重要意义。

4. 遇到的困难及解决方法

无