OSLab4 OpenHarmony 基础

PB22151796 莫环欣

实验目的

- 学习如何编译鸿蒙内核镜像
- 学习如何向开发板烧录镜像
- 学习如何对代码进行交叉编译并在不同平台上使用
- 学习如何为鸿蒙内核添加一个新的子系统
- 学籍如何使用 miniconda 进行环境隔离
- 学习如何使用 HDC 与开发板进行交互

实验环境

- 编译环境： Ubuntu 20.04.4 LTS
- 烧录环境： Windows 11
- 开发板： DAYU200

实验步骤

编译

首先，下载源码并解压，接着下载预编译内容后创建预编译文件夹并解压进去，随后进入源码根目录运行预编译下载脚本。

完成以上两步后，若预先没有安装 miniconda，需要先安装，并建立一个`python3.8`环境，并将其设置为常驻。

随后，进入内核源码根目录为`build`文件夹安装`hb`，下载/更新所有需要的库文件之后，就可以使用`hb`设置我们将要编译的系统并进行编译

下面就是一个包含了所有所需操作的脚本文件

> 注意，测试中发现不能直接执行，下载会出错

```
# 下载源码
cd ~/oslab
wget -O oh4.0_mini.tar.zst https://git.ustc.edu.cn/ldeng/ustc-os-resources/-/raw/
main/oh4.0_mini_no_kernel_20240423.tar.zst

echo "3243ee90de1eead40a70156a47992c07d657f5a7f739a955820a37a0e2370a5f oh4.0_mini.tar.zst"
 | sha256sum --check

echo plz check if any problem
echo Enter to continue
echo Ctrl and C to quit
read dummy_variable0

unzstd -c oh4.0mini.tar.zst | tar xvf -

 # 下载预编译内容
wget https://git.ustc.edu.cn/ldeng/ustc-os-resources/-/raw/
main/oh4.0_mini_prebuilts_20240423.tar.gz

echo "957af04fab034d646c4a20aaf2e35f46ce96b219d5f8d824373907374ae38b21
oh4.0_mini_prebuilts_20240423.tar.gz" | sha256sum --check

echo plz check if any problem
echo Enter to continue
echo Ctrl and C to quit
read dummy_variable1

mkdir OpenHarmony-v4.0-Release/openharmony_prebuilts

tar xvf oh4.0_mini_prebuilts_20240423.tar.gz
 --directory=OpenHarmony-v4.0-Release/openharmony_prebuilts

cd OpenHarmony-v4.0-Release/OpenHarmony

./build/prebuilts_download.sh

# 安装miniconda
# wget https://mirrors.bfsu.edu.cn/anaconda/
miniconda/Miniconda3-latest-Linux-x86_64.sh
#
# bash Miniconda3-latest-Linux-x86_64.sh -b
#
# ~/miniconda3/bin/conda init
#
# source ~/.bashrc
#
```

```
# # 建立python环境
# conda create -y --name oh python=3.8 --no-default-packages
# conda activate oh   # 激活对应环境
# python -V
#
# echo 请将以下两行添加至 ~/.bashrc 的末尾
# echo conda activate oh
# echo export PATH=~/.local/bin:$PATH
# read dummy_variable2


cd ~/oslab/OpenHarmony-v4.0-Release/OpenHarmony
python3 -m pip install --user build/hb

hb help

echo plz check if any problem
echo Enter to continue
echo Ctrl and C to quit
read dummy_variable3

sudo apt update
sudo apt install build-essential flex bison ruby
libgl1-mesa-dev libglib2.0-dev libncurses5-dev libxcursor-dev
libxrandr-dev libxinerama-dev libexpat1-dev default-jdk libelf-dev
ccache libssl-dev genext2fs liblz4-tool curl libstdc++-11-dev


# 编译
hb set -p rk3568_mini_system
hb build --gn-args load_test_config=False --ccache=False
```

添加子系统

在编译上面源码的过程中，我们可以同时添加子系统

- 首先确定自己需要添加的模块（必须）与子系统（可等待后续慢慢添加）的名称
- 将其添加到源码根目录的`build/subsystem_config.json`文件中，例如

  ```
  ```json
  sample": {
     "path": "sample",
     "name": "sample"
   },
  ```
  ```

- 之后，项目的框架应该如下(没有的文件先创建，后续再修改)

```json
 sample
 └── simpleshell
     ├── BUILD.gn
     ├── bundle.json
     ├── include
     │   └── simpleshell.h
     └── src
         └── simpleshell.c
```

- 接着，将之前编写好的`simpleshell.c`文件放入框架中
- 同时，还需额外编写一个头文件，应有以下内容
  - 头文件包含防护
  - 兼容`extern`
  - 函数声明

```
#ifndef SIMPLESHELL_H
#define SIMPLESHELL_H
#ifdef __cplusplus
#if __cplusplus
extern "C" {
#endif
#endif

int split_string(char *string, char *sep, char **string_clips);
int exec_builtin(int argc, char **argv, int *fd);
int process_redirect(int argc, char **argv, int *fd);
int execute(int argc, char **argv);

#ifdef __cplusplus
#if __cplusplus
}
#endif
#endif
#endif  // SIMPLESHELL_H
```

- 接着，编辑`bundle.json`文件，添加部件描述

```
{
  "name": "@ohos/simpleshell",
  "description": "A subSystem with simple shell",
  "version": "3.1",
  "license": "Apache License 2.0",
  "publishAs": "code-segment",
  "segment": {
    "destPath": "sample/simpleshell"
  },
  "repoAddr": "https://gitee.com/openharmony/applications_app_samples.git",
  "dirs": {},
  "scripts": {},
  "component": {
    "name": "simpleshell",
    "subsystem": "sample",
    "syscap": [],
    "features": [],
    "adapted_system_type": ["mini", "small", "standard"],
    "rom": "10KB",
    "ram": "10KB",
    "deps": {
      "components": [],
      "third_party": []
    },
    "build": {
      "sub_component": ["//sample/simpleshell:simpleshell"],
      "inner_kits": [],
      "test": []
    }
  }
}
```

- 编辑`BUILD.gn`文件，添加编译规则

```
import("//build/ohos.gni")
ohos_executable("simpleshell") {
    sources = [
        "src/simpleshell.c"
    ]
    include_dirs = [
        "include"
    ]
    cflags = []
    cflags_c = []
    cflags_cc = []
    ldflags = []
    configs = []
    deps = []
    part_name = "simpleshell"
    install_enable = true
}
```

- 最后，修改`vendor/hihope/rk3568_mini_system/config.json`文件，在其文末仿照其它部件添加我们的新部件

完成添加过程后，直接进入源码根目录运行上文脚本中的最后两步即可

```
之所以先进行一次编译，是因为后续修改只需重新编译3000+项
完整源码需要编译11800+项
```

交叉编译

进入`Ubuntu`环境，首先确保已经安装下面的包

```
sudo apt update
sudo apt install gcc-aarch64-linux-gnu
```

然后将需要编译的代码使用下面的形式编译

```
aarch64-linux-gnu-gcc -static -o simpleshell simpleshell.c
```

编译好的可执行文件发送到主机，即可进行最后的测试环节

烧录 & 测试

连接开发板后直接运行下面的`bat`文件

需要注意的是，打开`RKDevTool`之后，需要先把编译好的镜像烧录到开发板上

首先右键`loader`项的空白块导入配置，接着再单击各项空白处，手动把所有的镜像文件导入对应位置。

全部导入完成后，按下开发板的`RESET`键与上一行中间的按键，待显示改变后松开`RESET`键，等再次显示后即可松开按键并点击`执行`进行烧录，待开发板发出响声（或看到显示烧录完成）后即表示烧录完成。

使用下面的文件，烧录完成后按两下回车继续执行，并按照输出的提示完成即可

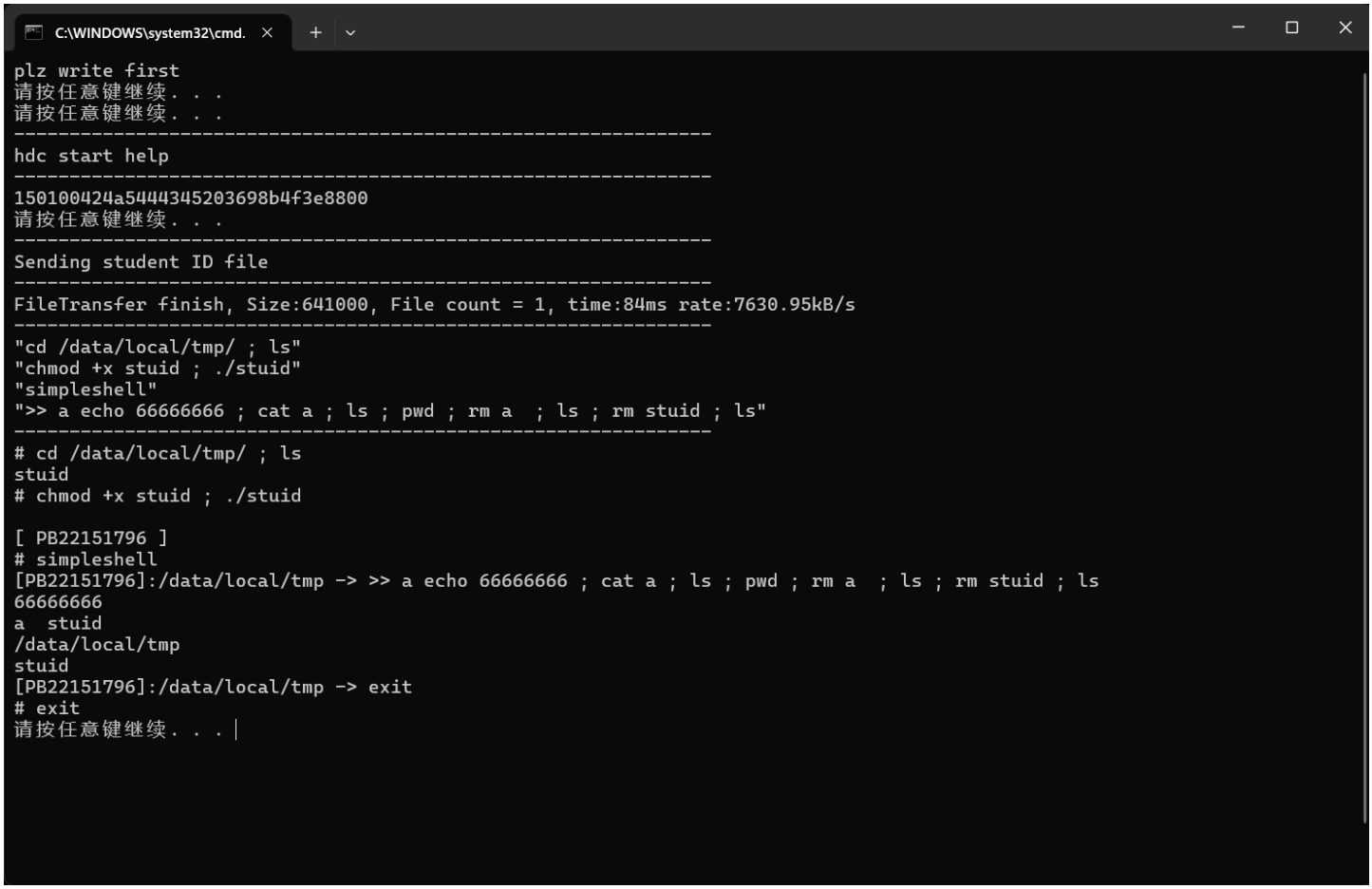
```
@echo off
echo plz write first
"D:\......\OS\OSLab\Lab4\tools\tools\windows\RKDevTool.exe"
pause
pause
echo ----------------------------------------------------------------
echo hdc start help
echo ----------------------------------------------------------------

hdc list targets
pause
echo ----------------------------------------------------------------
echo Sending student ID file
echo ----------------------------------------------------------------
hdc file send D:\......\OS\OSLab\Lab4\ID_part\stuid /data/local/tmp/
echo ----------------------------------------------------------------
echo "cd /data/local/tmp/ ; ls"
echo "chmod +x stuid ; ./stuid"
echo "simpleshell"
echo ">> a echo 66666666 ; cat a ; ls ; pwd ; rm a  ; ls ; rm stuid ; ls"
echo ----------------------------------------------------------------
hdc shell

pause
```

至此，实验结束

结果展示



可以看到我们已经成功在开发板上执行了交叉编译的程序并进入了子系统，实验成功

> 这里运行的批处理文件就是上面给出的