实验二 基于OpenWrt的网络应用软件开发

ZYX WJY ZHJ

读者须知:

1. 笔者所有步骤均在WSL上执行, Ubuntu版本为22.04-LTS, OpenWrt版本为24.10.0 x86_64, 不保证其他情况下也能顺利完成环境配置(不过经过试验, VM Ware完成网络配置后也能按照下面的步骤顺利执行).
2. 建议弄清每步的基本含义, 根据实际情况更改命令.
3. 与路径相关的命令一定记得更改.
4. 遇到有包或工具缺失一般直接下载即可, 如cmake.
5. 如有错误, 请多包涵, 也欢迎指正.

1 在Ubuntu上进行交叉编译

笔者在WSL安装的OpenWrt是基于x86架构的, 因此以x86平台为例下载SDK进行交叉编译.

大多数路由器并不是x86架构, 若要烧制程序到路由器可能需要更换SDK版本.

1.1 前置条件

set for Ubuntu 22.04 (that has older Python 3.xx), 其他版本见官方文档:

```
sudo apt update
sudo apt install build-essential clang flex bison g++ gawk \
gcc-multilib g++-multilib gettext git libncurses-dev libssl-dev \
python3-distutils python3-setuptools rsync swig unzip zlib1g-dev file
wget
```

下载SDK并解压

```
wget
https://downloads.openwrt.org/releases/24.10.0/targets/x86/64/openwrt-
sdk-24.10.0-x86-64_gcc-13.3.0_musl.Linux-x86_64.tar.zst
sudo apt install zstd
tar --zstd -xvf openwrt-sdk-24.10.0-x86-64_gcc-13.3.0_musl.Linux-
x86_64.tar.zst
# 重命名，此步可不选
mv openwrt-sdk-24.10.0-x86-64_gcc-13.3.0_musl.Linux-x86_64 openwrt-sdk-
24.10.0
```

1.2 配置环境变量

这段配置可能与使用本地gcc编译的其他项目有冲突, 建议编写为脚本文件, 执行后仅在当前终端生效, 而不是添加到shell配置文件中. 另外可能存在环境变量冲突, 建议先退出虚拟环境(如conda)再添加环境变量.

```
# 以下内容写入env.sh
# >>> openwrt >>>
export STAGING_DIR="/home/huaijin/my-code/openwrt-project/openwrt-sdk-24.10.0/staging_dir"
export TOOLCHAIN_DIR="$STAGING_DIR/toolchain-x86_64_gcc-13.3.0_musl"
export PATH="$TOOLCHAIN_DIR/bin:$PATH"
export TARGET=x86_64-openwrt-linux-musl
export CC=$TOOLCHAIN_DIR/bin/${TARGET}-gcc
export AR=$TOOLCHAIN_DIR/bin/${TARGET}-ar
export RANLIB=$TOOLCHAIN_DIR/bin/${TARGET}-ranlib
# <<< openwrt <<<
```

添加环境变量`source env.sh`. 不是`./env.sh`(仅被添加到执行脚本的子进程中, 而不是添加到当前终端的环境变量).

可以查看交叉编译器的位置`which x86_64-openwrt-linux-musl-gcc`, 有正常输出.

1.3 配置libpcap

使用SDK平台安装总是有很多依赖项缺失, 不如下载源码编译来得方便.

```
wget https://www.tcpdump.org/release/libpcap-1.10.4.tar.gz
tar -xzf libpcap-1.10.4.tar.gz
cd libpcap-1.10.4

./configure --host=$TARGET --with-pcap=linux --prefix=$STAGING_DIR/target-x86_64_musl/usr
# 最后一行有类似config.status: executing default-1 commands的输出

make
# 最后一行有类似chmod a+x pcap-config的输出

make install
# 最后一行有类似/`echo $i | sed 's/.manmisc.in/.7/'`; done的输出
```

1.4 创建项目

项目结构:

- openwrt-projects
  - project1
    - src
      - main.c
      - xxx.c
    - CMakeLists.txt
  - project2
  - toolchain.cmake

toolchain.cmake:

```
set(CMAKE_SYSTEM_NAME Linux)
set(CMAKE_SYSTEM_PROCESSOR x86_64)

set(OPENWRT_SDK "/home/huaijin/my-code/openwrt-project/openwrt-sdk-24.10.0")

set(STAGING_DIR /home/huaijin/my-code/openwrt-project/openwrt-sdk-24.10.0/staging_dir)

# 设置交叉编译器
set(CMAKE_C_COMPILER ${STAGING_DIR}/toolchain-x86_64_gcc-13.3.0_musl/bin/x86_64-openwrt-linux-gcc)
set(CMAKE_CXX_COMPILER ${STAGING_DIR}/toolchain-x86_64_gcc-13.3.0_musl/bin/x86_64-openwrt-linux-g++)

# 设置sysroot
set(CMAKE_FIND_ROOT_PATH ${STAGING_DIR}/target-x86_64_musl)

set(CMAKE_FIND_ROOT_PATH_MODE_PROGRAM NEVER)
set(CMAKE_FIND_ROOT_PATH_MODE_LIBRARY ONLY)
set(CMAKE_FIND_ROOT_PATH_MODE_INCLUDE ONLY)
```

CMakeLists.txt:

```
cmake_minimum_required(VERSION 3.10)
project(lab2)

set(CMAKE_C_STANDARD 11)

```

```
include_directories(
        ${CMAKE_FIND_ROOT_PATH}/usr/include
)

file(GLOB SOURCES src/*.c)

add_executable(lab2 ${SOURCES})

target_link_libraries(lab2
        pthread
        ${CMAKE_FIND_ROOT_PATH}/usr/lib/libpcap.so
)
```

1.5 编译

Ubuntu编译调试: 在src下

- `sudo apt install libpcap-dev`
- `gcc *.c -Wall -pthread -lpcap -o lab2`

交叉编译在OpenWrt上运行的可执行文件: 在项目根目录下, 如project1

```
mkdir build
cd build
cmake .. -DCMAKE_TOOLCHAIN_FILE=/home/huaijin/my-code/openwrt-project/toolchain.cmake
make

file lab2
# 输出lab2: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib/ld-musl-x86_64.so.1, with debug_info, not stripped
```

完成后将可执行文件传输到OpenWrt.

2 OpenWrt上运行可执行程序

```
opkg update
opkg install libpcap # 安装的仅仅是运行时
chmod u+x lab2
./lab2
```