山东大学 计算机科学与技术 学院

操作系统 课程实验报告

学号：201705130120

实验题目：实验二

实验学时：2

实验目的：

该实验在目录 lab2 中完成。
（1）熟悉 Nachos 的 makefiles 的结构；
（2）熟悉如何在几个 lab 文件目录中构造相应的 Nachos 系统；

实验环境：ubuntu18 x64 windows10 clion python3

实验步骤：

项目代码：https://github.com/Yuandiaodiaodiao/nachos-cmake-x64

由于 makefile 实在是太不人性化 本次实验将展示如何把 Makefile 的 nachos 改造为 cmake 的 nachos 并完成自动化构建
首先说明 c++编译的流程
源代码（source coprede）→预处理器（processor）→编译器（compiler）→汇编程序（assembler）→目标程序
对应于 nachos 的 makefile 文件
先通过 gcc 从源文件(.cc / .s)生成到.o 文件
其中.cc 文件直接被 gcc 编译为.o 文件 .s 文件先根据 switch.h 进行宏替换,然后使用 as 编译为.o 文件
在需要的.o 文件都编译好之后 一同送入 linker 产生可执行文件 由于使用静态链接 所以无需生成库文件
将.o 文件进行 link 之后即可生成可执行文件
可以看到是标准的 c++编译流程
那么理论上来说可以将其转换成 cmake 项目

首先 因为同时存在.cc 和.s 要让 cmake 执行汇编文件的编译流程 需要开启

```
ENABLE_LANGUAGE(ASM)
```

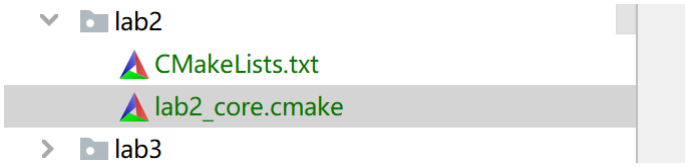
然后 由于需要在 x64 平台上编译 32 位程序 则需要配置编译参数

```
set(CMAKE_CXX_FLAGS "-m32")
set(CMAKE_ASM_FLAGS "-m32")
```

然后观察 makefile 在每个项目里都在编译参数内添加了三个宏 **HOST_i386 HOST_LINUX CHANGED**
那么 cmake 中也要为这些宏添加定义

```
add_definitions(-DHOST_i386)
add_definitions(-DHOST_LINUX)
add_definitions(-DCHANGED)
```

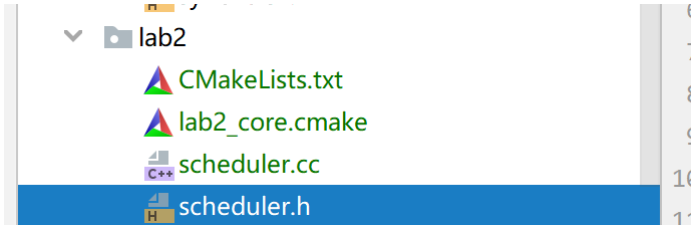
然后我们新建 lab2 的文件夹并添加



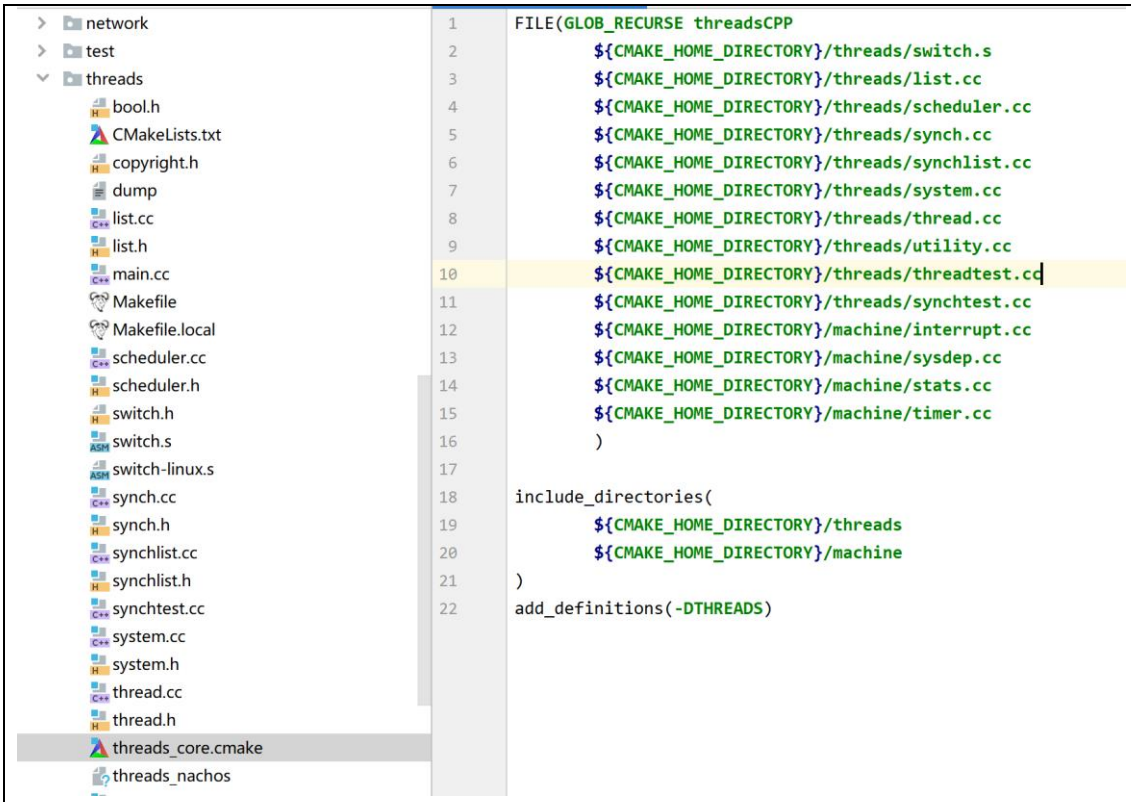
对应于 makefile 和 makefile.local
而同样的 makefile 里 include 了 makefile.local
我们在 CmakeLists.txt 里也要引入.cmake

```
include(lab2_core.cmake)
```

按照实验指导书的要求 我们要修改 scheduler.cc 需要在 lab2 里修改
那么新建一份并将.h 也一并拷贝过来



Makefile 把构建的项目输出到同目录下
Cmake 中通过 SET(**EXECUTABLE_OUTPUT_PATH ${CMAKE_CURRENT_SOURCE_DIR}**)
将文件输入到当前 cmakelists 的目录下
Lab2 需要包括 machine threads 的一些 cpp
所以在 threads 下新建



将必要的 cc 引入

然后在 lab2 中引入

```
include(${CMAKE_HOME_DIRECTORY}/threads/threads_core.cmake)
```

并且把当前目录也加入到 include 路径内

```
include_directories(.)
```

之后就可以构建可执行文件了

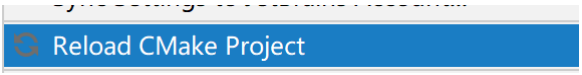
```
add_executable(lab2_nachos ./scheduler.cc ${threadsCPP} ${CMAKE_HOME_DIRECTORY}/threads/main.cc
```

将新加入的 **scheduler.cc 原本项目的 cc 文件集合 和 main.cc**

生成为可执行文件 lab2_nachos

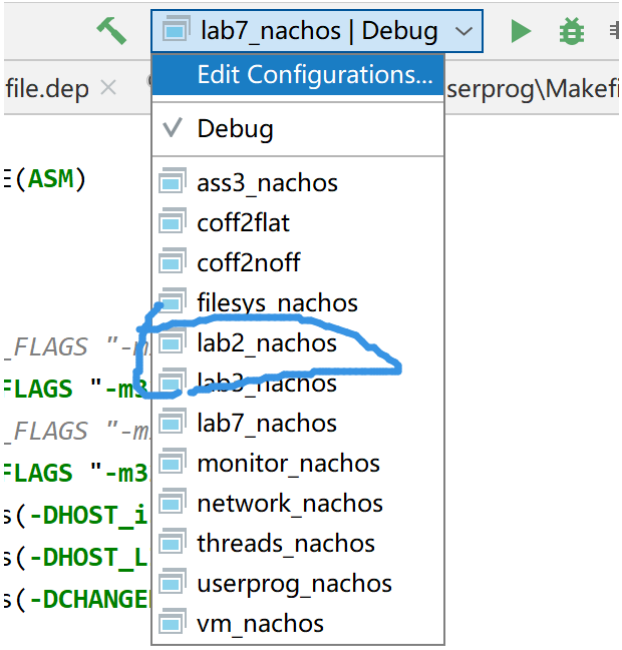
将 lab2 加入总的索引内

```
ENABLE_LANGUAGE(ASM)
# 32bit编译

#原有的警告
#set(CMAKE_CXX_FLAGS "-m32 -Wall -Wshadow")
set(CMAKE_CXX_FLAGS "-m32")
#set(CMAKE_ASM_FLAGS "-m32 -Wall -Wshadow")
set(CMAKE_ASM_FLAGS "-m32")
add_definitions(-DHOST_i386)
add_definitions(-DHOST_LINUX)
add_definitions(-DCHANGED)


add_subdirectory(threads)
add_subdirectory(ass3)
add_subdirectory(bin)
add_subdirectory(filesys)
add_subdirectory(lab3)
add_subdirectory(lab7-8)
add_subdirectory(monitor)
add_subdirectory(network)
add_subdirectory(userprog)
add_subdirectory(vm)
add_subdirectory(lab2)
```

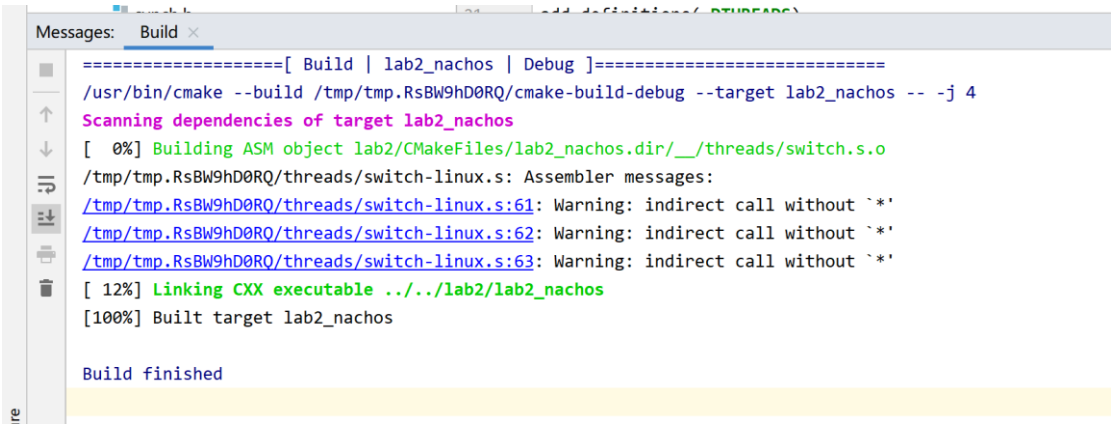


重新加载 cmake 项目

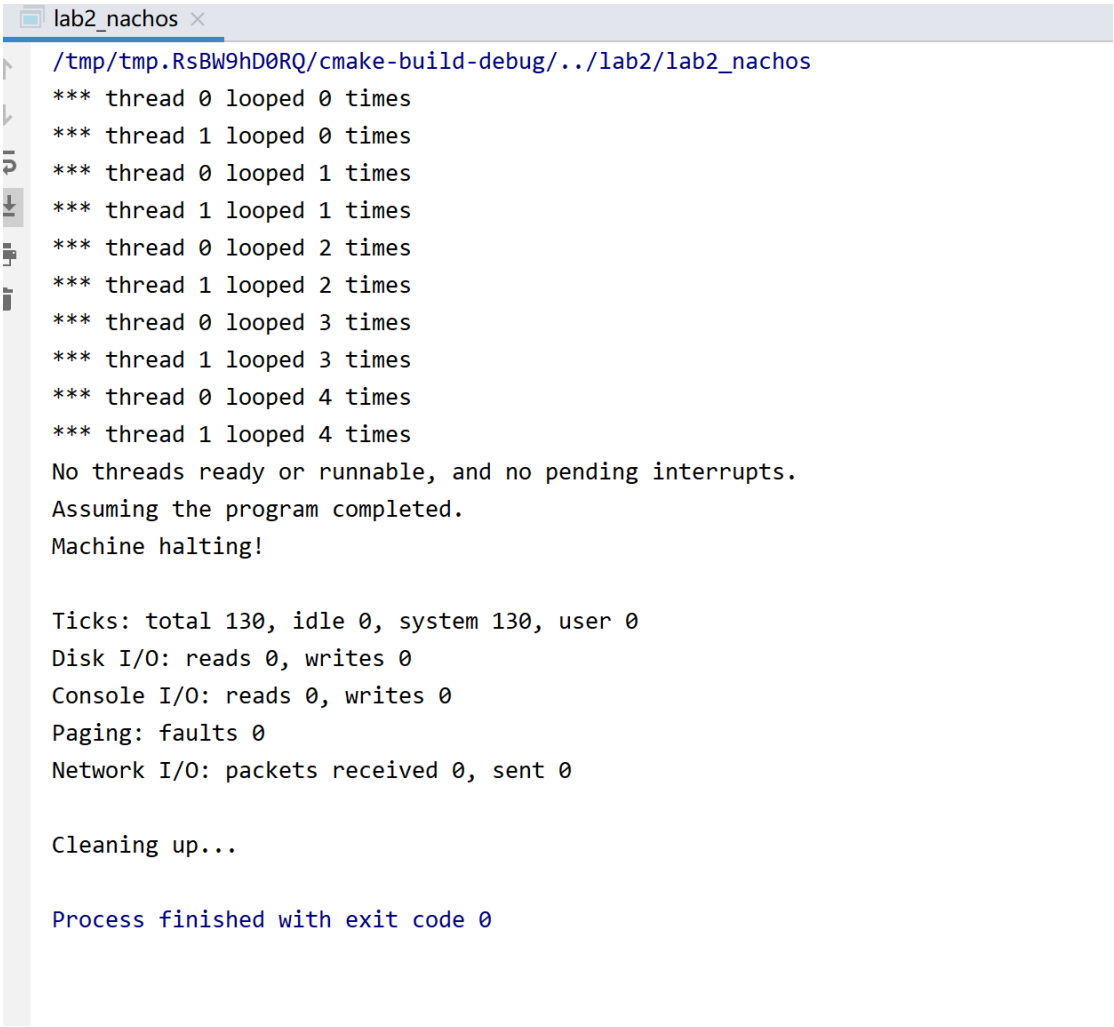
可以看到构建目标有了 lab2

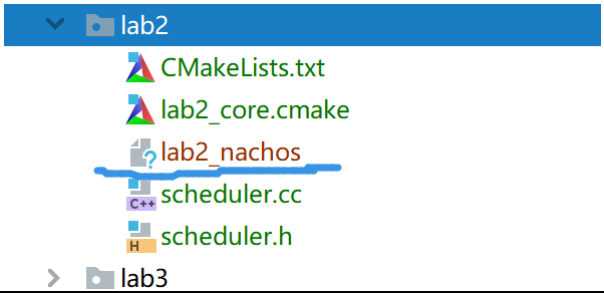


Build 项目


Messages: Build
====================[Build | lab2_nachos | Debug]============================
/usr/bin/cmake --build /tmp/tmp.RsBW9hD0RQ/cmake-build-debug --target lab2_nachos -- -j 4
Scanning dependencies of target lab2_nachos
[0%] Building ASM object lab2/CMakeFiles/lab2_nachos.dir/__/threads/switch.s.o
/tmp/tmp.RsBW9hD0RQ/threads/switch-linux.s: Assembler messages:
/tmp/tmp.RsBW9hD0RQ/threads/switch-linux.s:61: Warning: indirect call without `*'
/tmp/tmp.RsBW9hD0RQ/threads/switch-linux.s:62: Warning: indirect call without `*'
/tmp/tmp.RsBW9hD0RQ/threads/switch-linux.s:63: Warning: indirect call without `*'
[12%] Linking CXX executable ../../lab2/lab2_nachos
[100%] Built target lab2_nachos
Build finished

运行项目


lab2_nachos
/tmp/tmp.RsBW9hD0RQ/cmake-build-debug/../lab2/lab2_nachos
*** thread 0 looped 0 times
*** thread 1 looped 0 times
*** thread 0 looped 1 times
*** thread 1 looped 1 times
*** thread 0 looped 2 times
*** thread 1 looped 2 times
*** thread 0 looped 3 times
*** thread 1 looped 3 times
*** thread 0 looped 4 times
*** thread 1 looped 4 times
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!
Ticks: total 130, idle 0, system 130, user 0
Disk I/O: reads 0, writes 0
Console I/O: reads 0, writes 0
Paging: faults 0
Network I/O: packets received 0, sent 0
Cleaning up...
Process finished with exit code 0


lab2
CMakeLists.txt
lab2_core.cmake
lab2_nachos
scheduler.cc
scheduler.h
lab3

可以看到生成了可执行文件

实验结果：

问题及收获：

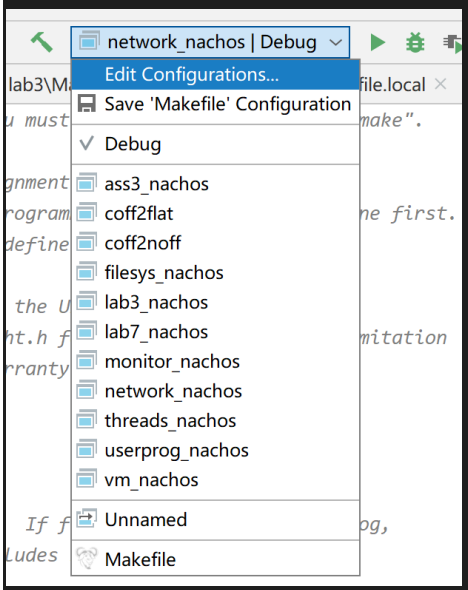
构建 cmake 项目心得

关于 cmake

test 文件夹暂时还是需要 makefile 的(与其说是 makefile 不如说是执行了一串脚本 改 cmake 也意义不大)

直接用 clion 打开 code 文件夹
如果你使用 macos/windows 请在 Toolchains 里配置 Remote 编译器

打开项目之后即可选择要构建的目标



如何使用

比如基础的 threads

内部有 cmakeList.txt 和 threads_core.cmake 你可以把他们类比为 makefile 和 makefile.local

threads_core.cmake 包含着基础的 cpp 路径和 include 路径

cmakeList.txt 内部通过 add_executable 来构建目标可执行文件

```
include(threads_core.cmake) 引入 cpp 和头文件
SET(EXECUTABLE_OUTPUT_PATH ${CMAKE_CURRENT_SOURCE_DIR}) 把可执行文件输出到当前目录下
add_executable(threads_nachos ${threadsCPP} main.cc) 添加 cpp 和 main 函数 生成的可执行文件为 threads
```

```
FILE(GLOB_RECURSE threadsCPP         将下列文件添加到 threadsCPP 的变量里
${CMAKE_HOME_DIRECTORY}/threads/switch.s
${CMAKE_HOME_DIRECTORY}/threads/list.cc         ${CMAKE_HOME_DIRECTORY}/threads/scheduler.cc
threads/thread.cc         ${CMAKE_HOME_DIRECTORY}/threads/utility.cc         ${CMAKE_HOME_DIRECTO
c
${CMAKE_HOME_DIRECTORY}/machine/stats.cc
${CMAKE_HOME_DIRECTORY}/machine/timer.cc         )
include_directories(
${CMAKE_HOME_DIRECTORY}/threads       添加包含目录
${CMAKE_HOME_DIRECTORY}/machine)
add_definitions(-DTHREADS) 添加宏定义
```

关于 make 变量定义 由 make 的 ifndef MAKEFILE_FILESYSdefine MAKEFILE_FILESYSyesendef

迁移到 cmake 风格的

```
set(MAKEFILE_FILESYS ON)
```

由 make 的 ifdef MAKEFILE_USERPROG_LOCALDEFINES := $(DEFINES:FILESYS_STUB=FILESYS)elseINCPATH +=

迁移到 cmake 风格的

```
if(MAKEFILE_USERPROG_LOCAL)
else()
    add_definitions(-DFILESYS_NEEDED -DFILESYS)
endif()
```

有关于

```
DEFINE:= $(DEFINES:FILESYS_STUB=FILESYS)
```

还没有弄懂

总 cmakelists.txt

在项目目录下有公共 list

```cmake
project(nathos) # 项目名称
cmake_minimum_required(VERSION 3.10)#  最低 cmake 版本
execute_process(COMMAND rm ${CMAKE_CURRENT_SOURCE_DIR}/threads/switch.s)  # 执行两行脚本用来对 swi
execute_process(COMMAND sh ${CMAKE_CURRENT_SOURCE_DIR}/build-switch.sh ${CMAKE_CURRENT_SOURCE_DI


#添加对.s 的编译
ENABLE_LANGUAGE(ASM)
# 32bit 编译

#原有的警告
#set(CMAKE_CXX_FLAGS "-m32 -Wall -Wshadow")
set(CMAKE_CXX_FLAGS "-m32")
#set(CMAKE_ASM_FLAGS "-m32 -Wall -Wshadow")
set(CMAKE_ASM_FLAGS "-m32")
add_definitions(-DHOST_i386)
add_definitions(-DHOST_LINUX)
add_definitions(-DCHANGED) # 这三个宏定义贯穿整个项目

# 下面是把每个子项目都包含进来 让 clion 能索引到
add_subdirectory(threads)
add_subdirectory(ass3)
add_subdirectory(bin)
add_subdirectory(filesys)
add_subdirectory(lab3)
add_subdirectory(lab7-8)
add_subdirectory(monitor)
add_subdirectory(network)
add_subdirectory(userprog)
add_subdirectory(vm)
```

```

学号姓名实验一.doc；
```