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	姓名：苑宗鹤	班级：17 1 班
实验题目：实验六		
实验学时：2	实验日期：2020/4/24	

实验目的：

（1） 为后续实验中实现系统调用 Exec()与 Exit()奠定基础

（2） 理解 nachos 可执行文件的格式与结构

（3） 掌握 nachos 应用程序的编程语法，了解用户进程是如何通过系统调用与操作系统内 核进行交互的

（4） 掌握如何利用交叉编译生成 nachos 的可执行程序

（5） 理解系统如何为应用程序创建进程，并启动进程

（6） 理解如何将用户线程映射到核心线程，核心线程执行用户程序的原理和方法

（7） 理解当前进程的页表是如何与 CPU 使用的页表进行关联的

实验环境：ubuntu18 x64 windows10 clion

实验步骤：

1 修改 coff.h 使 coff2noff 能够在 64bit 系统中运行

shell.c progtest.cc addrspace.h addrspace.cc coff.h

```
/* coff.h
 *   Data structures that describe the MIPS COFF format.
 */

#ifdef HOST_ALPHA       /* Needed because of gcc uses 64 bit long
#define _long int       /* integers on the DEC ALPHA architecture.
#else
#define _long int /* for x64 system int is _long*/
#endif
```

修改 cmake 文件重新编译两个文件

```
FILE(GLOB_RECURSE coff2noffCPP
        coff2noff.c

        )
FILE(GLOB_RECURSE coff2flatCPP
        coff2flat.c

        )
include_directories(
        ${CMAKE_HOME_DIRECTORY}/threads
        ${CMAKE_HOME_DIRECTORY}/bin
)

SET(EXECUTABLE_OUTPUT_PATH ${CMAKE_CURRENT_SOURCE_DIR})
add_executable(coff2noff ${coff2noffCPP})
add_executable(coff2flat ${coff2flatCPP})
```

生成之后要给生成的可执行文件添加执行权限

```
$ chmod +x ../bin/coff2noff
$ make clean
```

```
$ chmod +x ../bin/coff2flat
```

修改 test/makefile

```
coff2noff = ../bin/coff2noff
coff2flat = ../bin/coff2flat
```

Noff.h 中

包含了 NOFFMAGIC 的定义

数据结构 noffHeader

包含三项 Segment 类型的 code,initData,uninitDate 一个代码段两个数据段

每个段由 segment 中的 virtualAddr inFileAddr size 组成 分别表示了在文件中的地址 内存中的地址 和大小

```
yuan@ubuntu:/tmp/tmp.RsBW9hD0RQ/test$ vi halt.s

        .file   1 "halt.c"
gcc2_compiled.:
__gnu_compiled_c:
        .text
        .align  2
        .globl  main
        .ent    main
main:
        .frame  $fp,24,$31              # vars= 0, regs= 2/0, args= 16, extra= 0
        .mask   0xc0000000,-4
        .fmask  0x00000000,0
        subu    $sp,$sp,24
        sw      $31,20($sp)
        sw      $fp,16($sp)     保存栈空间开辟新栈
        move    $fp,$sp
        jal     __main
        jal     Halt            main函数
$L1:                            调用halt
        move    $sp,$fp
        lw      $31,20($sp)
        lw      $fp,16($sp)     弹出栈 恢复sp
        addu    $sp,$sp,24
        j       $31
        .end    main
~
~
~
```

观察指令

```
arch  halt.c  halt.flat  halt.noff  halt.s  Makefile  matmult.c  matmult.flat  matmult.no
yuan@ubuntu:/tmp/tmp.RsBW9hD0RQ/test$ vi halt.s
yuan@ubuntu:/tmp/tmp.RsBW9hD0RQ/test$ cd ..
yuan@ubuntu:/tmp/tmp.RsBW9hD0RQ$ cd userprog/
yuan@ubuntu:/tmp/tmp.RsBW9hD0RQ/userprog$ ./userprog_nachos -d m  -x ../test/halt.noff
*** 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
Starting thread "main" at time 120
At PC = 0x0: JAL 52
At PC = 0x4: SLL r0,r0,0
At PC = 0xd0: ADDIU r29,r29,-24
At PC = 0xd4: SW r31,20(r29)
At PC = 0xd8: SW r30,16(r29)
At PC = 0xdc: JAL 48
At PC = 0xe0: ADDU r30,r29,r0
At PC = 0xc0: JR r0,r31
At PC = 0xc4: SLL r0,r0,0
At PC = 0xe4: JAL 4
At PC = 0xe8: SLL r0,r0,0
At PC = 0x10: ADDIU r2,r0,0
At PC = 0x14: SYSCALL
Exception: syscall
Machine halting!

Ticks: total 132, idle 0, system 120, user 12
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...
yuan@ubuntu:/tmp/tmp.RsBW9hD0RQ/userprog$
```

输出 page 表

```
pagetable dump:9pages
V    P
0 0
1 1
2 2
3 3
4 4
5 5
6 6
7 7
8 8
9 9
Starting thread "main" at time 12
```

理解 nachos 为应用程序创建进程的过程

-x 时执行 startProgress 创建进程

理解系统为用户进程分配内存空间、建立页表的过程，分析目前的处理方法存在的问题

通过 AddrSpace 对象来为用户进程分配内存空间

通过读取.noff 文件头来分配用户程序空间和页表项，创建页表对象后将地址空间清零，再将数据段和代码段放入内存

理解应用进程如何映射到一个核心线程

通过将 space 指向用户进程的地址切换

实验结果： 仓库地址： https://github.com/Yuandiaodiaodiao/nachos-cmake-x64
问题及收获：