NACHOS PROJECT ASSIGNMENT 3

PART 1

CS 370 – OPERATING SYSTEMS, LUMS

"THIS ASSIGNMENT HAS BEEN ADAPTED FROM CS162 COURSE @WASHINGTON-UNIVERISTY"

USERLEVEL PROGRAMS & MULTIPROGRAMMING

DUE DATE: REFER TO INDIVIDUAL TASKS*

* Slips not applicable on non-programming tasks.

'YOU SHOULD READ THE PROJECT MECHANICS DOCUMENT BEFORE STARTING WORK ON THIS ASSIGNMENT.'

"NACHOS HAS BEEN DEVELOPED BY TOM ANDERSON, WE ONLY REUSE THE SOURCE"

-- All the submissions will be checked for plagiarism - do not resort to any unfair means --

Overview

Congratulations on making it through Programming Assignment 2. You are now the proud owner of working operating system that has been partially coded by you, yourself. ☺ However, there is some little but challenging work left before you can actually run multiple real C-language programs on your NachOS. In this part of your CS370 assignment, we will be introducing "system calls" and making our OS support multiprogramming.

As in the second assignment, we provided you some of the code you need; your job is to complete the system and enhance it. However, before we can get started with the real job, we need some preparation.

Background & Preparation (90)

Up to now, all the code you have written for Nachos has been part of the operating system kernel. In a real operating system, the kernel not only uses its procedures internally, but also allows user-level programs to access some of its routines via "system calls".

These user level programs link themselves with the OS at runtime. What does this mean?

Imagine you write a C-programfor a simple calculator with only +, -, *, / operations. You compile it and run it to see the calculator on your screen. Now, if you decide to add more functionality to the calculator like support for mod, trigonometric functions (sine, cosine, etc.) you will have to close the program, write more code, recompile and rerun the program. You don't just write new code that the calculator picks up on the run to change its functionality.

Operating Systems are software too, just like the calculator - but you don't want to shut down your OS every time you add some new functionality in the form of new software – Here is where linkers come in!

TASK 1 - LINKERS (30) **DUE: Nov. 11, 2015**

THIS TASK IS TO BE DONE INDIVIDUALLY – EVEN INTRA-GROUP COLLOBORATION IS NOT ALLOWED.

Search Linkers on Google or Bing– submit a 200-400 word summary on what linkers are and how linker works (especially run-time linkers) – Make sure you understand these concepts well, spend at least 1-2 hours on related readings – DONOT RELY ON WIKIPEDIA ALONE.

Submit your solutions on LMS **(SLIP NOT APPLICABLE, EMAIL SUBMISSIONS WILL NOT BE ENTERTAINED, SUBMISSIONS WILL BE COMPARED WITH ONLINE RESOURCES FOR PLAGIARISM)**

The code we provide for this assignment already implements load-time linking and can very easily (with few lines of code) be extended to allow for run time linking. However, the constraint is with understanding how linking works in NachOS, and how custom c-programs are loaded and executed.

TASK 2 - UNDERSTANDING LINKING AND CODE EXECUTION IN NACHOS (60) **DUE: Refer to sub parts**

Every operating system has its own executable/object file format (windows has .exe files). For NachOS we have NOFF (NachOS Object File Format). This file is generated using a predefined function named **COFF2NOFF***. What COFF2NOFF does is simple: it takes a COFF(C Object File Format) file and converts it to a NOFF file. NachOS loads and runs this NOFF file (like windows loads and runs .exe files). Note: NachOS does not support floating point operations.

* **COFF2NOFF** program is found in the **nachos-3.4/code/bin** directory. You can run it by typing **./COFF2NOFF** in the **bin** folder.

Part A – Understanding how a C-program is converted to NOFF in NachOS (20) **DUE: Nov 11, 2015**

Go to **nachos-3.4/code/test** and open **halt.c** (type: **nano halt.c** or **gedit halt.c**). Have a look: this program simply calls a "system call" Halt() – which terminates the program and NachOS itself too.

You task is to understand how this C-program is converted first to COFF and then to NOFF at compile time.

Hint1: The file named **halt** (without any file extension) in the **/code/test** folder is a NOFF file.
Hint2: You can run the halt program on NachOS by going to the **userprog** directory and typing:
./nachos -x ../test/halt
Hint3: Go to **nachos-3.4/code/test** and examine the following code in **Makefile**:

```
halt.o: halt.c

	$(CC) $(CFLAGS) -c halt.c
halt: halt.o start.o
	$(LD) $(LDFLAGS) start.o halt.o -o halt.coff
	../bin/coff2noff halt.coff halt
```

Hint4: The compiled programs must be linked with some special flags (as can be seen above), then converted into Nachos format, using the program "coff2noff". - Google and Bing these flags.

Submit a 100-200 word summary on how NOFF files are being generated from .c files.

Submit your solutions on LMS **(SLIP NOT APPLICABLE, EMAIL SUBMISSIONS WILL NOT BE ENTERTAINED, SUBMISSIONS WILL BE CHECKED FOR PLAGIARISM)**

In this assignment we are giving you a simulated CPU that models a real CPU. In fact, the simulated CPU is the same as the real CPU (a MIPS chip), but by simulating the execution, we have complete control over how many instructions are executed at a time, how the address spaces work, and how interrupts and exceptions (including system calls) are handled.

Part B – Understanding how a NOFF file is loaded and executed (40): DUE: Nov 11, 2015

This part requires an understanding of Part A.

Go to the userprog directory (**nachos-3.4/code/userprog**) and run **make**. Now open the NachOS executable in a visual debugger environment (Emacs – GDB), ensure that tunneling is enabled and Xming is already running.

Set a break point at the first line of the main file and then type **run nachos –x ../test/halt** in the gdb console. NachOS is now running and it has been told to load halt from the test directory. Note: **–x** tells nachos to load a program. Also note that **../test/halt** is the path to the halt file (it is the NOFF file) we want to run.

Now trace the execution path (till termination) to get a good picture of how it is working.

Tip 1: Read the intro comments of each file (.h and .cc) that you visit while tracing the execution path.

Tip2: Relevant code is in the following directories: userprog, machine, test, filesys (only 2 files: filesys and openfile) –For short intro of files relevant to this assignment refer to appendix 1.

Tip 2: Discuss with peers. Discuss again….

You are to submit a flow chart for this question. The flow chart should explain how the nachos get to know that it needs to load a file and which file needs to be loaded. The flow chart should give a clean and concise view how the flow transfers from reading the NOFF file -> loading it to memory -> executing it.
