NACHOS PROJECT ASSIGNMENT 4

CS 370 - OPERATING SYSTEMS

RUNNING USERLEVEL PROGRAMS ON NACHOS

DUE DATE: DECEMBER 10, 2015 (11:59PM)

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

THIS PROJECT HAS BEEN ADAPTED FROM THE NACHOS PROJECT OF THE CS 162 COURSE AT WASHINGTON UNIVERSITY

All will be checked for plagiarism - do not attempt to use any unfair means. All cases of unfair means use will be referred to the Disciplinary Committee.

Overview

You now have your own multiprogramming enabled operating system. The system is capable of loading and running a multiple user program at run time. We now want to test your operating system and see if it does everything it is supposed to.

This assignment consists of 4 questions; each requires you to write a 'C' program that you will run using the shell program given in the Nachos code/test/ directory. Marks will be awarded on functionality; essentially meaning that we will be following a binary marking scheme where you get full marks if the C program works on your nachos, otherwise you get a zero. **You get no marks if your C programs are correctly written but your nachos implementation is unable to run them.**

Note: To allow for plagiarism check, we have saved a snap shot of the current states of each group's code. For the final submission (i.e. this assignment), each group is expected to run the code on the server – those using personal machines are requested to inform the TA's by email.

Assignment (100)

This assignment will use the Nachos functionality we developed over the semester. It will not take you more than 2-3 hours to complete this assignment if your NachOS is in a properly working state. However, you will not be able to properly run programs you make in this assignment, unless you have properly implemented threading, system calls and multiprogramming – consider this assignment to be a second chance to identify and correct any flaws in your system. Doing well in this assignment will make up for any deficiencies in previous assignments.

Help note: Place each program you make in **/code/test/ directory.** Recall that you will need to add a new entry in the **Makefile** of the **test** folder to compile a **noff** file for your new program.

1. I eat memory and do nothing (25)

Write a memory hungry program that initializes an array of 5000 integers; it then prints 'I eat memory and do nothing' and exits properly using the exit system call.

This question will get credit if we are able to run this program multiple times on shell without restarting nachos. Make sure that the exit system call properly frees the memory space allocated to the program.

2. Student Sort (25)

Write a program that reads '**CS370Students.csv**' and writes a new file, '**Sorted-CS370Students.csv**' with the records sorted on student names alphabetically. Use the system calls you made earlier. Do not try using fstream library.

You get credit for this task if your program manages to successfully create the '**Sorted-CS370Students.csv'** file.

3. Word Count1(25)

Write a program that takes a Word (string) as input from the console and outputs (on the console) the number of instances of the input word in the '**CS370WordList.csv**' file.

4. Word Count2(25)

Write a program that reads a file '**CS370SomeText.csv**' and writes a new file, '**WordCount-CS370SomeText.csv**' with each unique word written along with its word count in the text.

Appendix 1 (from assignment 3)

The new code is spread over several directories. There are some new kernel files in “userprog”, there are a few additional machine simulation files in “machine”, and a stub file system in “filesys”. The user programs are in “test”, and utilities to generate a Nachos loadable executable are in “bin”. Since Nachos executes MIPS instructions (and there aren't very many MIPS machines left!), we also provide you a cross-compiler. The cross-compiler runs on Linux and compiles user programs into MIPS format.

You may find Narten’s "road map" to Nachos helpful; Google it. Also, it is OK to change the constants in "machine.h", for example, to change the amount of physical memory, if that helps you design better test cases (you may choose not to). The files for this assignment include:

USERKERNEL.H, USERKERNEL.CC -- routines for booting and testing a multiprogramming kernel.

ADDRSPACE.H, ADDRSPACE.CC -- create an address space in which to run a user program, and load the program from disk.

SYSCALL.H -- the system call interface: kernel procedures that user programs can invoke.

EXCEPTION.CC -- the handler for system calls and other user-level exceptions, such as page faults. In the code we supply, only the “halt” system call is supported.

BITMAP.H, BITMAP.CC -- routines for manipulating bitmaps (this might be useful for keeping track of physical page frames)

FILESYS.H, OPENFILE.H (found in the filesys directory) -- a stub defining the Nachos file system routines. For this assignment, we have implemented the Nachos file system by directly making the corresponding calls to the UNIX file system; this is so that you need to debug only one thing at a time.

TRANSLATE.H, TRANSLATE.CC -- translation table routines. In the code we supply to run “halt”, we assume that every virtual address is the same as its physical address -- this restricts us to running one user program at a time. You will generalize this to allow multiple user programs to be run concurrently. We will not ask you to implement virtual memory support until in assignment 3; for now, every page must be in physical memory.

MACHINE.H, MACHINE.CC -- emulates the part of the machine that executes user programs: main memory, processor registers, etc.

MIPSSIM.H, MIPSSIM.CC -- emulates the integer instruction set of a MIPS R2/3000 processor.

CONSOLE.H, CONSOLE.CC -- emulates a terminal device using UNIX files. A terminal is (i) byte oriented, (ii) incoming bytes can be read and written at the same time, and (iii) bytes arrive asynchronously (as a result of user keystrokes), without being explicitly requested.

SYNCHCONSOLE.H, SYNCHCONSOLE.CC -- provides synchronized access to the console device.