CS 460 Programming Assignment One Linux Tools, System Calls

Due Feb 8, 2016, 11:59pm Email me your SVN repository name
Due Feb 12, 2016, **4:45pm** Final Project (in class demo!) **50 points**

Goal:
Refresh your skills on: Eclipse, Subversion, the Linux command line, C, Makefiles, argv/argc, and learn about Linux System Calls and some Linux command line tools.

Description:
You will write a program that will make a number of system calls to determine information about the system your program is running on. You'll need to display this information to the screen as specified below. Further, and most important, you will need to run your program using **`strace`** and **`ltrace`** to investigate how Linux is executing your program. You will need to submit answers to the questions at the end of this handout. The answers must be written in full English sentences in a text file named **Answers.txt** in the root of your Eclipse project.

Your Program:

The system calls you will need to invoke are:
nanosleep()
access()
sysinfo()
uname()

> You can learn more about these system calls by typing **man systemcall** (without the ()) at the Linux command line or **Googling 'man systemcall'**

You will also need to use errno/perror() to find and display error messages produced by the system calls. You will need to determine which header files supply each of the above functions.

Your program must do the following:

1. Pause for 1.5 seconds using nanosleep
2. Call **uname** and display all of the returned information to the screen using printf.
3. Call **access** using F_OK to determine if you have access to the file passed as a command line argument (argc/argv). Use **perror**() to print an error if you don't have access.
4. Call **access** to determine if you have **read** access to the file passed as a command line argument. Use **perror**() to print an error if you don't have access.
5. Call **access** to determine if you have **write** access to the file passed as a command line argument. Use **perror**() to print an error if you don't have access.
6. Call **access** to determine if you have both **read** and **write** access to the file passed as a command line argument. Use **perror**() to print an error if you don't have access.
7. Call **sysinfo** with NULL as the argument and handle the error using **perror**.
8. Call **sysinfo** and display all of the returned information to the screen (separated by \n). You must turn the load information into a float. The load numbers should be very close (with rounding) to what is displayed by **top**.
9. You must check to make sure exactly one file name is passed as a command line argument. If this is not the case you need to print a usage message and terminate.
10. You must write at least one function (referred to as **foo**, but don't name it that) other than main().
    Error handling: For nanosleep, check for errors and then decode the error using errno.
    For all other system calls, check for errors and then use perror() to display an error message.
    NOTE: Using perror writes to stderr so your error output may not be in the order you expect or the order on this handout!

Sample Output:

```
coffee$ ./CS460_SysCalls /etc/passwd

nanosleep ------------------------------------

UNAME ----------------------------------

Linux | coffee | 3.11.10-29-default | #1 SMP Thu Mar 5 16:24:00 UTC 2015 (338c513) |
x86_64 | (none)

access File ----------------------------------

access Read  ----------------------------------

access Write  ----------------------------------

/etc/passwd with W_OK: Permission denied

access Read and Write  ----------------------------------

/etc/passwd with R_OK | W_OK: Permission denied

sysinfo NULL  ----------------------------------

sysinfo(NULL): Bad address

sysinfo  ----------------------------------

uptime: 520768
Loads: 0.361816 0.480957 0.495117
Total Ram: 16804556800
Free Ram: 1811079168
Shared Ram: 0
Buffer Ram: 818171904
Total Swap: 8585736192
Free Swap: 8585736192
Processes: 763
Total High Memory: 0
Free High Memory: 0
Memory Unit Size: 1
```

NOTE: Your numbers/data will vary!

Usage Output:

```
coffee$ ./CS460_SystemCalls
USAGE: ./CS460_SysCalls filename
```

Subversion (or how do I submit my work?):

You must store your source code in a Subversion repository on zeus. You need to create on as follows:

zeus$ svnadmin create /home/***login***/SVNREPOS_CS460S16/

You can connect to this through Eclipse using the address:
svn+ssh://***login***@zeus.cs.pacificu.edu/home/***login***/SVNREPOS_CS460S16/

Name your project **CS460_SysCalls_PUNetID**. I will check out your code using the following command:

zeus$ svn co -r {2016-02-12T16:50} svn+ssh://zeus.cs.pacificu.edu/home/***PUNetID***/SVNREPOS_CS460S16/ CS460_SysCalls_***PUNetID***

This will pull out the last revision made previous to 4:50pm on Feb12, 2016.

Makefile

You will need to build a Makefile for this Eclipse project. You need the following make targets:

```
CS460_SysCalls: build the executable file CS460_SysCalls (using -g)
runTest:        run './CS460_SysCalls /etc/passwd'
runNoFile:      run './CS460_SysCalls'
runStrace:      run 'strace ./CS460_SysCalls /etc/passwd > strace.out 2>&1'
runLtrace:      run 'ltrace ./CS460_SysCalls /etc/passwd'
runTime:        run 'time ./CS460_SysCalls /etc/passwd'
clean:          remove any executable and object files
```

I will compile your code (on Zeus) using the command:

```
zeus$ gmake clean ; gmake
```

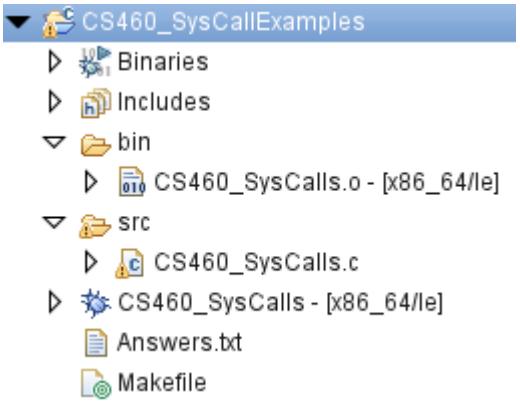
I will test your code using the command (among others):

zeus$ gmake runTest

Eclipse

Build a C Project | Makefile Project | Empty Project | Linux GCC toolchain.

Your Eclipse project must look very similar to the project shown here. Note the executable file and Answers.txt are at the root of the project. You may add an **include** directory. You must adhere to proper Computer Science department coding standards!



Useful resources

http://linux.die.net/man snprintf fflush(stdout)

Questions:

Answer the following questions in the file Answers.txt in your Eclipse Project. Use proper English, complete sentences, and sensible formatting. Put your name at the top of the file.

1. How many hours did you spend coding this project?

2. Which part of this project caused you the most difficulty?

3. What did you need to do to convert the load averages from sysinfo into a float? How did you determine how to do this?

4. Given the output of runTime, does nanosleep seem accurate?

5. Study the output of **`ltrace ./CS460_SysCalls /etc/passwd.`**
   1. Does the call to **main** show up? If so where, if not, why not?
   2. Does the call to the **foo** show up? If so where, if not, why not?
   3. What does **access("/etc/passwd", 2) = -1** mean? Where does the **2** come from? What does the **-1** mean?
   4. Do you see calls to **puts**? Did you call **puts**? Why do you think **puts** is there?

6. Study the output of **`strace ./CS460_SysCalls /etc/passwd > strace.out 2>&1.`**
   1. To do this, open the file strace.out in Eclipse. You may need to refresh the project for the file to show up in the Project Explorer (press **F5**).
   2. Why is Linux trying to open libc.so? What directories are looked at first? Why are the directories in that order? Where is libc.so eventually found?
   3. What is happening between the open and close of libc.so? Can you determine what each function call is doing? **I do not expect you to decode every function call here. But give it your best effort. readelf -a helps!** I do expect you to be able to identify the open and close of each file.
   4. What are execve() and brk(0) doing?
   5. Where does `write(2, "/etc.....` come from? Where does the 2 come from?
   6. Where does write(1, "access....." come from? Where does the 1 come from? How many times does write(1 appear? Does this make sense?

7. Use Eclipse to navigate <time.h> to determine how time_t is implemented. List every file, line number pair you go through to get to the ultimate definition.
   Hint: you should end up at a #define using primitive types (int, long, float, double, char, void)
   Hint: by putting your cursor on a header file name, datatype, or function and pressing F3 inside Eclipse you will be taken to that item's definition.
   Hint: Declare a variable of type time_t and use F3 to start looking for its definition. Your first step should take you to time.h. Investigate how __time_t is defined. Recurse until you hit a line containing only primitive types.

8. The answers to these questions are the most important part of this assignment!

☺ Bring a printout of strace.out and your answers to class on Friday 12, 2016.

This took me less than 200 non-comment lines of code. The most time consuming part of this assignment is the Questions section and the **research** you will need to do. I expect questions to come up during class before the assignment is due.