CS 460 Assignment #1 **Linux Tools, System Calls**

Date assigned: Wednesday, February 5, 2014
Date due: Wednesday, February 12, 2014 **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:

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

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

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 | 2.6.34.10-0.4-desktop | #1 SMP PREEMPT 2011-10-19 22:16:41 +0200 |
x86_64 |

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: 2412082
Loads: 0.084473 0.062988 0.073730
Total Ram: 16598081536
Free Ram: 340078592
Shared Ram: 0
Buffer Ram: 617562112
Total Swap: 2154819584
Free Swap: 2154786816
Processes: 740
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. If you do not already have a repository on zeus you need to create one as follows:

zeus$ svnadmin create /home/***login***/SVNCS460REPOS/

You can connect to this repository through Eclipse using the address:
svn+ssh://***login***@zeus.cs.pacificu.edu/home/***login***/SVNCS460REPOS/

Name your project **CS460_SysCalls_PUNetID**.

On the day your assignment is due, you are to submit a zipped up tar file on zeus called **CS460_SysCalls_PUNetID.tar.gz** and then using the submit script, type the following:

```
zeus$ submit cs460s14 CS460_SysCalls_PUNetID.tar.gz
```

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 - Wall)
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
```

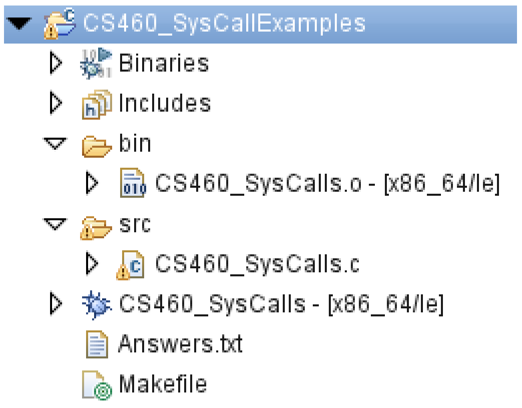
I will compile your code (on Zeus) using the command:

```
zeus$ make clean ; make
```

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

```
zeus$ make 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. Each answer is to be preceded by the question identifier (e.g. 6. b) 1.). 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.`
   a) Does the call to **main** show up? If so where, if not, why not?
   b) Does the call to the **foo** show up? If so where, if not, why not?
   c) Consider **access("/etc/passwd", 2) = -1**
      1. What does this system call mean?
      2. Where does the **2** come from?
      3. What does the **-1** mean?
   d) 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.`
   a) 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**).
   b) What are each of the following system calls doing at the very beginning of the program? Be as specific as you can but don't be incorrect with what you say. i.e. more is not necessarily better if it's wrong.
      1. `execve ("./CS460_SysCalls", ...)`
      2. `brk(0)`
      3. `mmap (NULL, 4096, …)`
      4. 4096 is a length (or number of bytes). 4096 is not magical. Why 4096 and not say 4000 or 1024?
   c) Why is Linux trying to open libc.so?
      1. What directories are looked at first?
      2. Why are the directories in that order?
      3. Where is libc.so eventually found?
   d) Where does `write(2, "/etc......"` come from?
   e) Where does the 2 come from in `write(2, "/etc......"` ?
   f) Where does `write(1, "access....."` come from?
   g) Where does the 1 come from in `write(1, "access....."`?
   h) 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 line 76. Investigate how __time_t is defined. Recurse until you

hit a line containing only primitive types.

8. In the folder /home/CS460/2014 on zeus, you will find an executable called debug. The program produces a Segmentation fault. Explain exactly why.

The answers to these questions are the most important part of this assignment! Do not copy and quote any text from a reference. All answers are to be explained in your own words.

Bring a printout of strace.out and your answers to class on Wednesday February 12, 2014.

This took me less than 200 non-commented 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.