Operating Systems **Homework # 1 – Chapters 1 & 2**

Written answers (questions 1-3, and 5) are to be submitted in a single file, preferably Microsoft Word (txt or pdf files are also okay). Name this file hw1.docx (or .txt or .pdf) and submit using moodle.

Question 4 asks you to write a small program. You should submit only your source file, named p1.c. This is also to be submitted by using moodle.

Remember, you MUST put your name at the top of both files (i.e. a comment block at the top of your hw1.cpp file).

1) Explain the difference between a *program* and a *process*. **(10 points)**

2) Dual-Mode Execution (kernel vs. user mode) is an important feature of modern operating systems. The purpose of having a separate kernel and user mode is to prevent user programs from executing dangerous instructions.

    a) Describe how the CPU determines the current execution mode (kernel vs. user mode) **(5 points)**

    b) What prevents a (malicious/poorly designed) user program from elevating itself to kernel mode? **(5 points)**

3) The book describes three different ways that parameters can be passed to the operating system when a system call is made.

    a. List the three methods. **(5 points)**

    b. Why aren't system call parameters passed the same way parameters are passed to any other function (i.e. through the call stack)? **(5 points)**

4) Write a program that prints the integers from 0 – 10,000 to stdout. Keep track of how much time your program spends executing both user and system instructions. Have your program print out these time values. In addition, have your program print out the operating system version, machine name, release level, and hardware platform it is running on (print this information *after* the 10,000 integers). **(15 points)**

For example, running the program on my machine resulted in the following output:

```
1
2
…
9999
10000
Operating System Version:  Darwin
Machine Name:  sfrees.local
Release Level:  11.4.2
Version Level:  Darwin Kernel Version 11.4.2: Thu Aug 23 16:25:48 PDT 2012; root:xnu-
1699.32.7~1/RELEASE_X86_64
Hardware Platform:  x86_64
User time:  0.0073870000
System time:  0.0143600000
```

You will be using two POSIX calls for this program:

uname (stuct utsname *): the **uname** function allows you to retrieve information about the system your program is running on. The **uname** function takes one parameter - a reference to a **utsname** structure. After **uname** is called, this data structure will be filled in with the relevant system information. (You must include sys/utsname.h to use this function)

getrusage (int who, struct rusage *): the **getrusage** function allows you to retrieve information regarding the running of your program. You must include sys/resource.h to use this function. The **getrusage** function takes two parameters:

who: integer identifying the program(s) you are trying to find information about. You should use **RUSAGE_SELF** for this program, a predefined constant telling **getrusage** to get information about the current/calling program only.

rusage: a reference to an **rusage** data structure. After **getrusage** is called the **rusage** data structure will be filled in. Along with other important information, the **rusage** data structure contains information about how much time your program has spent executing user and system instructions. Remember, **rusage** will give you the time in a **timeval** structure, which contain seconds **and** microseconds.

***In order to complete this program you will likely need to look up additional information about these functions and data structures. A good place to start is the UNIX man pages (type "man" and the term you are looking for at the command line on phobos). You will also find a wealth of information about these functions on the web (i.e. try typing "man rusage" into Google)*

5) Add a call to the sleep function [sleep (int seconds)] before you print out the system information. Make your program sleep for 5 -10 seconds. Run your program with and without the sleep call. ***How does sleep effect the time spent executing user and system instructions? Why don't the resulting times reported by rusage add up to the amount of time it took your program to complete?*** You **do not** need to turn in the source code/program, just include your answers in the document containing the answers to questions 1-4. **(5 points)**