Assignment 1

Part I – Questions – This part must be done on your own. Each question is worth 20 points.

1. There are two ways to communicate with device registers. One for the hardware to associate device registers with memory locations and to use the typical load and store instructions to access them. This method of communicating with device registers is referred to as memory-mapped input/output (I/O). The second method is to provide a separate device address space called port-space I/O. In this case, separate instructions (IN/OUT in Intel architectures) that function like load and store except that they read/write numbered device registers are used.

   Thinking about concepts that we have covered in this unit, explain how hardware could prevent user programs from accessing the device registers for
   a. address-mapped device registers
   b. port-space I/O.
2. Suppose two central processing unit (CPU) cores have separate caches. If cache write through is not enabled, will we see cache inconsistency or cache incoherency? Briefly justify your answer.
3. A web commerce application P_A launches 3 processes:
   - P_D inventory database
   - P_W web server
   - P_B business to business inventory manager

   The web server (P_W) launches the following processes:

   - P_C credit card authorization service
   - P_U user credential authentication service

   Draw a process tree for P_A. What will the operating system do to each of these processes if P_W were to unexpectedly terminate (e.g. reference to invalid memory).
4. Describe what happens when a virtual machine executes a privileged instruction. How should this behavior differ when the virtual machine is executing user mode vs. privileged mode instruction? How does the virtual machine know the difference?

Part II – Get your feet wet with UNIX/C/C++ (60 points)
Unless otherwise specified, programming assignments may be done with a pair programming partner following the specified methodology in the Williams and Kessler (2001) article which is assigned reading for all students regardless of whether or not you are pair programming.

This is a small programming assignment designed to give you experience in making system calls, reading manual pages, and writing makefiles which are a way to manage compilation. This program must execute correctly on systems implementing the POSIX standard (e.g. edoras which runs CentOS, or any modern linux distribution).

Directories are actually special files that contain information about other files. In this programming assignment, you will get your feet wet with system calls and manual pages. In this assignment, you will write your own version of the "ls" command which lists files in a directory. Your version of ls, myls will be significantly simpler than the standard ls.

Basic functionality when user executes myls:

1. If no command line arguments are given, the current directory is listed with a newline after each file including the last one. By default, any file starting whose first character is "." will not be listed *unless the hidden flag is set* (see below).
2. An optional flag, -h (for hidden), may be specified which will list all files or directories including those that start with a ".".
3. One or more command line arguments may be given. Each argument is a directory to list, i.e. "myls a02 a03" should list the files in directories a02 and a03. If any of the specified directory files cannot be accessed, the error "Cannot access a02" should be printed.
4. You must create a makefile (see course FAQ) such that when someone types "make" in your working directory it will compile the program with an output of myls.
5. When you submit, be sure to disable any debugging code that you might have included, we will be comparing your output against expected output.

You will need the system commands, opendir, closedir, and readdir to complete this assignment. For details on how to use these, you would normally use UNIX's on-line manual pages to learn about the functions signature (parameters and return code), but they have not yet been installed on Edoras. Fortunately, there is a https://www.kernel.org/doc/man-pages/. You may find it easier to use getopt to parse the -h flag, but this is not required (it will be on a future assignment). If you are confused by the function signatures in the manual pages, see the FAQ entry on interpreting function signatures (prototypes).

What to turn in:

Both parts I and II must be submitted to Canvas separately. Remember Part I must be done individually. For Part II, you will need to ensure that you add your pair programming partner if applicable (you will have an opportunity to do this). Submit source files, a makefile that successfully compiles your program when someone types make, and a demo of your program running in file output.txt. Do not submit executables, git repositories, etc.