SY204 – Sys Prgm & OS Fund Name(s): _____

Alpha(s): _____

Assignment Type:	Programming – Laboratory	Collaboration Policy:	Default
Assignment Title:	0x02: Hiding from `ls`		

1. General.

Due	
Portion of Assignment	Course Server Time / Date
Pseudo Code	Friday, 1800 24 Feb 2017
Complete	Beginning of Class Tuesday, 28 Feb 2017

a. Description. Read through this entire assignment before starting work. In this assignment you will create a program that first creates a file in the file system on disk and then removes that file from the file system on disk but continues to use the file within the kernel's internal representation of the file system.

2. Explore (20).

a. [2 / __ / 0] Read the `ls(1)` man page, and answer the following questions.

What option to `ls` displays the i-node number (index) of the file system entity? _____

What option to `ls` displays and sorts output by the last access time? _____

What option to `ls` displays and sorts output by the last modified time? _____

b. [2 / __ / 0] Read the `ln(1)` man page, and answer the following questions.

What is the default type of link that `ln` creates? _____

What option to `ln` creates a symbolic link? _____

[Remainder of Page Intentionally Blank]

c. (10) Perform the following steps in order, answer the questions in the order listed. Enter the following commands in an Ubuntu 16.04 LTS x86-64 terminal session.

```
[101]$ mkdir explore
[102]$ cd explore/
[103]$ mkdir foo
[104]$ mkdir foo/bar
[105]$ mkdir bar
[106]$ echo hello world > hello.txt
[107]$ ls -lai
```

[2 / __ / 0] Section 1.

What character does `ls` use to indicate a directory? _____

What character does `ls` use to indicate a regular file? _____

What is the reference count for the current working directory? _____

What is the reference count for `foo/`? _____

What is the reference count for `hello.txt`? _____

What is the i-node for `hello.txt`? _____

```
[108]$ cat hello.txt
[109]$ ln hello.txt hello2.txt
[110]$ ln -s hello.txt hello3.txt
[111]$ cat hello2.txt
[112]$ cat hello3.txt
[113]$ ls -lai
```

[2 / __ / 0] Section 2.

What is the reference count for `hello.txt`? _____

What is the reference count for `hello2.txt`? _____

What is the i-node for `hello2.txt`? _____

What is the reference count for `hello3.txt`? _____

What is the i-node for `hello3.txt`? _____

[2 / __ / 0] Section 3.

What character does `ls` use to indicate a symbolic link? _____

Creating a hard link creates a new entry in the i-node Table. True / False

Creating a symbolic link creates a new entry in the i-node Table. True / False

```
[114]$ rm -f hello.txt
[115]$ cat hello2.txt
[116]$ ls -lai
```

What is the reference count for `hello2.txt`? _____

```
[117]$ cat hello3.txt
```

[2 / __ / 0] Section 4.

Describe why the previous command fails.


```
[118]$ rm -f hello2.txt
[119]$ ls -lai
```

[2 / __ / 0] Section 5.

Describe when an i-node and the data blocks associated with an i-node are actually removed from the file system on disk, use terms associated with an i-node.

You may remove the remaining files and directories you created.

d. [2 / __ / 0] Read the `link(2)` man page, read the `symlink(2)` man page, read the `unlink(2)` man page, and answer the following questions.

The result of a call to `link` is similar to which command line from Part 2.c? _____

The result of a call to `symlink` is similar to which command line from Part 2.c? _____

Which command from Part 2.c makes use of the `unlink` system call? _____

e. [2 / __ / 0] Read the `mkstemp(3)` man page and answer the following question.

In your own words describe what risk there is with using `mkstemp`.

f. [2 / __ / 0] Read the `lseek(2)` man page, and answer the following questions.

Scenario: You want to reset the offset of a sequence of bytes for a file descriptor back to the beginning of the sequence of bytes using `lseek`.

What value should be passed in as `offset`? _____

What value should be passed in as `whence`? _____

[Remainder of Page Intentionally Blank]

3. Plan (20). You may attach additional pages as needed.

a. [6 / __ / 0] Draw a high-level block control flow diagram that depicts the ordering of the tasks of Implement Part 1 and Part 2. The Hard Coded Pause Points may serve as a good separation.

b. [6 / __ / 0] Based on your block control flow diagram, write pseudo code for the `main` function in your program. Write the pseudo code in `lsHidingPseudo.c`. Include a comment that clearly indicates your pseudo code labeled *Paragraph 3.b*.
Note: Your pseudo code will not be compiled, you do not need to write your pseudo code such that it compiles. Your pseudo code *shall not* match your final solution code. Go through the problem solving process for programming.
Hint: Comments are your friend.

c. [3 / __ / 0] Take one of your code blocks from above and drill down into that code block. Depict the system calls that you will need to use within the code block and their relationships (their ordering). Write pseudo code in `lsHidingPseudo.c`. Include a comment that clearly indicates your pseudo code labeled *Paragraph 3.c*.

d. [2 / __ / 0] List the options to the Descriptor I/O function that you plan to specify when initially accessing the new randomly named file.

e. [3 / __ / 0] Given that there is a way for a program in execution to actually start another program, in your own words describe how the features specified in Implement Part 2 can be used in an operational scenario. In other words what capability does Implement Part 2 add to our tool bag?

[Remainder of Page Intentionally Blank]

4. Implement (30).

a. [0 / -5 / -10] Sound and Secure Cyber Design

- You shall minimize the use of global variables other than `errno`, and for error reporting.
- You are encouraged to use `#define`, and other preprocessing directives.
  - Your program will not be compiled using the `-d` compiler directive.
- Source code is poorly or inconsistently formatted, but can be compiled (readability).
- Source code uses poorly named functions or poorly named variables (readability).
- Source code does not use associated common libraries (reusability).
- Source code does not use library defined constants (portability).
- Source code does not use deprecated language features (design for the future).
- Implementation does not violate Principle of Least Privilege.

b. [3 / __ / 0] Usage Option (All Parts)

- Read in the "`-h`" (help) option using `getopt`, that does not take any arguments.
- Upon detection of the "`-h`" option display simple usage information that describes how to use your program. Example: program name, listing and discussion of command line arguments.
- You may assume that all options will appear before arguments.
- If no command line arguments are passed in, output the usage message.
- You may use Stream I/O (`printf`) for this feature.
- After you output the usage message, exit the program with a normal return value.

c. [3 / __ / 0] Error Handling (All Parts)

- For an error that occurs from a system call, output a simple error message (recall `perror(3)`), and exit the program returning an error number.
- At least return a different error number for each of the systems calls that you use. That is, multiple calls to the same system call need not return unique error numbers, but calls to different system calls shall return unique error numbers.
- You may use Stream I/O for reporting errors.
- Output error messages to standard error.

d. [12 / __ / 0] Part 1 – Hiding from `ls`

- Use Descriptor I/O operations to open a file specified as a command line argument and read at least one byte from the sequence of bytes before Hard Coded Pause Point 1.
- Use file system operations to remove the i-node entry for the file opened in between Hard Coded Pause Point 1 and Hard Coded Pause Point 2. You may assume the file specified on the command line will have an on disk reference count of 1.
- Close Descriptor I/O operations to the open file after Hard Coded Pause Point 2.

Note: To correctly test this portion of the implementation you should perform the operations in (run your program from) a section of the file system that is not an NFS (Network File System); in a department UNIX lab environment your home directory (and subdirectories) are an NFS section of the file system. The `/tmp/` directory on any department environment is not an NFS section of the file system; `/tmp/` is typically not an NFS section on any UNIX system. Any user can create files in `/tmp/`, the temporary directory.
Note: If you are successful, the file that you specified as a command line argument will no longer be in the file system on disk. You should test this feature with that in mind.

e. [12 / __ / 0] Part 2 – A File by Any Other Name

- After Hard Coded Pause Point 2 but before you have closed Descriptor I/O operations, reset the buffer offset to the beginning of the sequence of bytes for the original file specified via the command line.
- Using Descriptor I/O open a new randomly named file on the file system with read, write, and *execute* permissions for the user.
  Note: Read the `tempnam(3)` man page. You may disregard the note to never use this function (for this assignment); you may disregard the compiler warning regarding `tempnam()` usage.
- Using Descriptor I/O copy the bytes from the original file specified via the command line to the new randomly named file. The original file specified will be an arbitrary length.
  Note: If you are confident of this feature specify your compiled program as the command line argument, then test by running the created copy.
- Implement at least the reading and writing steps as a function your program calls.

5. Test (30).

a. [0 / -15] Code Compilation and Running

- Source code compiles (`gcc -Wall ...`) under Ubuntu 16.04 LTS x86-64 with no warnings and no errors.
- Compiled program runs without crashing on test cases (no SEGFAULTs, no infinite loops).

b. [3 / __ / 0] Usage Option (All Parts)

- Solution correctly outputs a usage message when the help option is entered.
- Solution correctly outputs a usage message when no command line arguments are entered.
- Solution correctly returns a normal exit value.

c. [3 / __ / 0] Error Handling (All Parts)

- Solution correctly outputs error messages to standard error.
- Solution correctly returns unique values for different system calls.

d. [12 / __ / 0] Part 1 – Hiding from `ls`

- Solution correctly uses Descriptor I/O.
- Solution correctly hides from `ls`.
- Solution correctly removes the original copy of the file.

e. [12 / __ / 0] Part 2 – A File by Any Other Name

- Solution correctly creates a different named copy of the file.
- Solution correctly sets executable permissions on the created copy of the file.

6. Submit. Turn in this worksheet and submit source code per the following specifications.

- Turn in this worksheet to your instructor after you have submitted your final source code.
- Submit your pseudo code as a single file named `lsHidingPseudo.c`, following the submission directions on the course information web page.
- Submit your final code as a single file named `lsHiding-#.c`, where # is the part of the implementation that you completed (e.g. `lsHiding-2.c`), following the submission directions on the course information web page.
- `submit` Assignment (project): *lab02*

[0 / -10 / -15 / -20] Submitted source code that demonstrated insufficient effort (lack of attempt to solve problem)

[0 / -5] Submitted assignment using incorrect assignment type or incorrect assignment number.

[0 / -5] Submitted incorrect number of files in submission directory.

[0 / -5] Submitted file(s) named incorrectly.

[0 / -5] Submitted file(s) do not include name and alpha.