CSCI 4061: Introduction to Operating Systems (Spring 2014)

Released: Feb 18, 2014 Due: Mar 7, 2014

1 Pre-requisites

It is assumed that you have **good** knowledge of C. Defining good: you can write, compile, run and debug C code.

2 Introduction

This assignment will introduce you to file and directory operations in Unix. It will also teach you how to use pipes for communication between processes in Unix.

3 Problem

`grep` is one of the most useful commands in Unix. Using `grep` users can search input files for lines containing a particular string. In this assignment you will write a simple `grep`-like utility called `getstring` which will search all files in the input **directory tree** for lines containing a particular string. Since the input directory tree might contain sub-directories, `getstring` must traverse the entire directory tree starting from the input directory.Consider using `grep` to search for the input string. You can read the contents of a file and pipe it as input to `grep` along with the search string. **For this assignment, you are not allowed to pass the filename as an argument to `grep`.**

Input:

```
$ getstring <absolute-path-of-input-directory> <search-string>
```

`getstring` is the name of the executable. There will be exactly one directory and one search string specified as inputs to `getstring`.

Output:
The program should print all the lines that contain the search string from all the files in the input directory to an output file named `out.txt`. It should also print the *absolute path* of every file and sub-directory in the order in which they are encountered to a logfile named `log.txt`.

Clarifications:

1. The input directory tree will have only text files and sub-directories.
2. The order in which lines appear in the output file and the names appear in the logfile will depend on how you traverse the directory tree. Any traversal order is acceptable.
3. The search string will be a simple string of characters. It will not specify a regular expression.
4. Do not make any assumptions on the sizes of the files in the input directory tree.
5. The absolute path of filenames will not exceed 256 characters.

4 Example:

Consider an input directory tree as shown below. The content of each file in the tree is indicated in [ ].

```
/root/input-directory-tree:
  file1.txt --> [red]
  file2.txt --> [red and blue]
  sub-dir-1:
    file3.txt --> [green and yellow]
    file4.txt --> [blue and yellow]
  sub-dir-2:
    file5.txt --> [red and yellow]
    file6.txt --> [yellow]
```

Assume `getstring` is run as:

```
$ getstring /root/input-directory-tree red
```

Then the contents of `out.txt` could be (this order might differ depending on how you traverse the directory tree):

```
red
red and blue
red and yellow
```

The contents of `log.txt` could be (this order might differ depending on how you traverse the directory tree):

```
/root/input-directory-tree/file1.txt
/root/input-directory-tree/file2.txt
/root/input-directory-tree/sub-dir-1
/root/input-directory-tree/sub-dir-2
/root/input-directory-tree/sub-dir-1/file3.txt
/root/input-directory-tree/sub-dir-1/file4.txt
/root/input-directory-tree/sub-dir-2/file5.txt
/root/input-directory-tree/sub-dir-2/file6.txt
```

5 Extra Credit:

Test inputs for extra credit will have hard and soft links apart from regular text files. These links might point to other files in the input directory tree. Therefore it is possible that `getstring` might encounter the same file multiple times while traversing the directory (either directly or through hard or soft links). In case of soft links `getstring` must ignore such links. For hard links `getstring` must make sure that the file is searched just once - when it is encountered the first time during traversal.

Clarifications:

1. There will be separate test cases for extra credit which will have hard and soft links apart from text files. Hard and soft links have to be considered only for the extra credit portion.
2. If your program handles links in the input directory tree, then the output file must also contain any lines in the linked files (hard links only) that contain the search string. It must also print the *absolute path* of all links (hard and soft), along with the names of other files and sub-directories, to the logfile.
3. Hard links and soft links will need to differentiated and handled appropriately.

6 Extra Credit Example:

Consider the previous example which has been modified to include links. The file pointed to by a link is indicated in { }.

```
/root/input-directory-tree:
  file1.txt --> [red]
  file2.txt --> [red and blue]
  soft-link1 --> {/root/input-directory-tree/sub-dir-2/file5.txt}
  sub-dir-1:
    file3.txt --> [green and yellow]
    file4.txt --> [blue and yellow]
  sub-dir-2:
    file5.txt --> [red and yellow]
    file6.txt --> [yellow]
    hard-link2 --> {/root/input-directory-tree/sub-dir-2/file5.txt}
```

Assume `getstring` is run as:

```
$ getstring /root/input-directory-tree red
```

Then the contents of `out.txt` could be (this order might differ depending on how you traverse the directory tree):

```
red
red and blue
red and yellow
```

The contents of `log.txt` could be (this order might differ depending on how you traverse the directory tree):

```
/root/input-directory-tree/file1.txt
/root/input-directory-tree/file2.txt
/root/input-directory-tree/soft-link1
/root/input-directory-tree/sub-dir-1
/root/input-directory-tree/sub-dir-2
/root/input-directory-tree/sub-dir-1/file3.txt
/root/input-directory-tree/sub-dir-1/file4.txt
/root/input-directory-tree/sub-dir-2/file5.txt
/root/input-directory-tree/sub-dir-2/file6.txt
/root/input-directory-tree/sub-dir-2/hard-link2
```

7 Hints:

1. You will need to use `opendir` to open the input directory. To iterate through the contents of the directory you will need to use `readdir`.
2. Consider using `read` and `write` system calls to handle files and pipes instead of `fopen` and `fclose`.
3. You have to use `stat` to check if an entry within a directory is a file or a sub-directory. For the extra credit portion, you can use `lstat` to check if an entry is a soft link.

8 Few Do's

Technical:

1. Program must be written in C.
2. Make sure your program compiles using gcc compiler on any of the CSELab ubuntu machines. We will use the same development Environment. No points will be given if program does not compile.
3. You must also provide a Makefile along with the source code. Please make sure that the name of the executable produced by the Makefile is *getstring*.
4. **Your program must produce only the expected output. Make sure all debugging output is cleaned before you submit your assignment.**

Non-Technical

1. Assignment can be done in group of at most two head counts. **Please include your names at the top in README file.**
2. **Please submit just one assignment per group.** Any one person from the group can make the submission as long as all names are included in the README file.
3. It is expected that you do not take any unfair advantage of any ambiguity in the problem statement. In that case, please confirm with Instructor or Teaching 0 before you proceed with any approach. It might not make sense coming up with a solution that does not help hone your skills in the expected area.

9 Submission Guidelines

The code and Makefile should be packaged and submitted as a tar.gz file. **Please do not use any other form of compression**. Name of the TAR ball should be **csci4061pa2**. This directory should contain the source files and the Makefile as described in section 8 above. Feel free to include any .c or .h files that you feel are necessary.

10 Grading

Maximum you can get for this assignment is 110%. Point distribution is as follows:

1. Following Submission Instructions and a sensible README:	5 points
2. Documentation and style:	15 points
3. Functionality and Error Handling:	80 points
4. Extra Credit:	10 points