Fall 2021 CSci 4061: Introduction to Operating Systems
Project #2 – Parallel String Matching
Instructor: Abhishek Chandra
Interim Submission Due: 11:59 pm, Oct. 20 (Wed.), 2021
Final Submission Due: 11:59 pm, Oct. 27 (Wed.), 2021

1. Background

Grep is a linux command that is used to find patterns in each file. It prints each line that matches the pattern. In this project you will be implementing “Parallel string matching” that will search for a string in all files and subdirectories of a root directory using multiple processes.

2. Project Overview

You will be using File I/O, Pipes, directory operations, links and I/O redirection. In this project you are required to spawn multiple child processes. Each child process is responsible for searching some portion of the root directory and communicates the search results to the parent using pipes. Parent is responsible for reading from these pipes and printing the output to the terminal. **Only the parent should print the output to the terminal.** For the final submission If you are using the child program to directly print to the terminal, none of the test cases will pass the grading script.

2.1. Input

PA2 program will take 2 arguments, the string to be searched and the path to the root directory. “String” is a single English word and “root directory” is any directory that needs to be searched.

```
Example : $ ./master <Input Directory path> <Word to search>
```

2.2. Output

The program prints all the lines containing the String. The output format is as follows:

[Path to the file containing the String] : Line containing the string

If there are multiple lines in a file containing the “String”, each such line is printed once.
Example :

```
$ ./master input/ip1 Imagine
input/ip1/Imagine_By_Lennon.txt: Imagine there's no heaven
input/ip1/Imagine_By_Lennon.txt: Imagine all the people
input/ip1/Imagine_By_Lennon.txt: Imagine there's no countries
input/ip1/Imagine_By_Lennon.txt: Imagine all the people
input/ip1/Imagine_By_Lennon.txt: Imagine no possessions
input/ip1/Imagine_By_Lennon.txt: Imagine all the people
```

2.3. String Matching

String matching needs to be case sensitive. Use of the **strstr()** function provided by the string library is strongly recommended. Autograding script will take into account the number of matches your program returned.

2.4. Child processes and Root Directory

The root directory may contain multiple first level subdirectories and files. One child process needs to be created for every first level subdirectory. This child process is responsible for searching all the files in its sub directories. A pipe must also be created for every child. Pipe is used by the child to communicate the results back to the parent.

Consider a possible directory structure as shown in **Fig 1**. Here the root directory ***/root*** contains 2 first level subdirectories, namely ***/root/f1*** and ***/root/f2***. Child 1 as shown in the figure, is responsible for searching files in ***/root/f1***. Child 2 as shown in the figure is responsible for searching files in ***/root/f2***.

Note that no new child is created for the ***/root/f2/f3*** directory. Child processes are only created for the first level directories. Also note that parent is responsible for searching first level files in this case ***File 5*** and no new child is created for this.

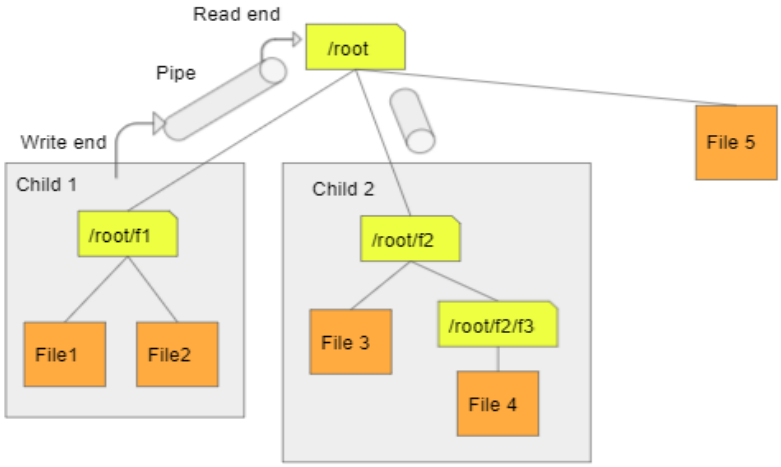

Fig 1: Input Directory structure

2.5. Pipes and I/O redirection

Pipe will be used by the child to communicate search results to the parent. Parent reads from the pipe and prints it to the console. Each child's STDOUT must be redirected to the write end of the pipe. One pipe must be created for each child process. In **Fig 1**, two pipes are created, one for each child.

2.6. Directory Traversal and Soft links

Each child process needs to traverse the sub-directory assigned to it recursively, searching every file in its directory sub structure and writing search results to the pipe. A file may be a regular file, link, directory etc. While traversing the directory, make sure to **not parse soft links files.**

Hint : Type of a file may be identified by **st_mode** present in the *stat* structure returned by ***lstat*** command.

2.7. Extra credit (Handling Hard Links)

A hard link is a directory entry that associates a name with a file in the file system. There can be multiple hard links pointing to the same file in the file system. For extra credit you need to identify hard linked files and parse the underlying file only once.

For example : Consider the directory structure in **Fig 2**, where two hard links, ***In1*** and ***In2*** point to the same file (***File***). You are required to parse ***File*** only once although you will encounter two entries to the same ***File*** during directory traversal.

Hint: This can be done using the inode number of the file present in stat structure returned by stat/lstat functions. You may also need to maintain a data structure that stores information about inode numbers already seen. **Important Note:** You may assume that hard links to a file are always present in the same first level subdirectory.

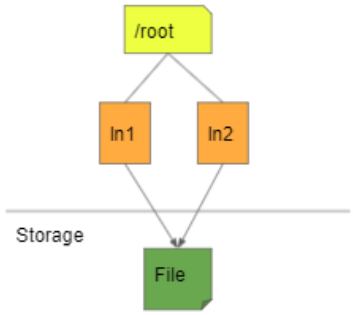

Fig 2 : Hard Links

3. Project Folder Structure

Please strictly conform with the folder structure. The conformance will be graded.
You should have a project folder named “PA2_G[Your group number]” (e.g. PA2_G23 for project group 23).
The project folder contains “include”, “lib”, “src”, “input” and “output” folders, as well as Makefile, README and two executables.

- “include” folder: .h header files
- “lib” folder: .o library files
- “src” folder: .c source files
- “input” folder: contains input folders
- “output” folder: Use $ make run to create output files for all test cases
- Two executables: master and child

4. Execution Syntax

The usage of pa2 is as follows.

4.1. Run master program directly by using :

$ ./master <path to the directory> <string to be searched>

4.2. Run test cases:

For test case 1: $ make run1
For test case 2: $ make run2 , and so on

4.3. Generate output:

$ make run

5. Assumptions and Hints

- You may assume that a pipe never gets full
- All hard links to a file are within a single first level directory
- strstr() function from string library may be used for string matching
- Maximum number of first level directories is 10
- No assumption can be made on depth of the directory
- Maximum file size is 5KB
- Maximum length of line is 1000 Bytes

6. Submission

6.1. Interim Submission

- Complete root directory traversal (in master.c) and create the required number of child processes.
- Child program should be launched with appropriate arguments using exec()
- Inside the child, print path to the directory assigned to it.
- Master should wait for all the child processes to complete.

  Interim submission should be a zip containing two .c files (master.c and child.c) and an Interim report (pdf). Interim report should include:
  - Project group number
  - Members' name and X500.
  - Plan on individual contributions
  - Captured screenshot showing terminal output for test case 4. If the requirements are correctly implemented, 7 child processes will be created.

Due date for the interim submission is Oct. 20, 2021 (Wed.), 11:59pm. No late submissions will be accepted.

6.2 Final Submission

One student from each group should upload to Canvas, a zip file containing the project folder. README should include the following details:

- Project group number
- Group member names and x500 addresses
- Members' individual contributions
- Any assumptions outside this document
- How to compile and run your program

The README file does not have to be long, but must properly describe the above points. Your source code should provide appropriate comments for functions. At the top of your README file and each C source file please include the following comments:

```
/*test machine: CSELAB_machine_name
* group number: G[Group Number]
* name: full_name1 , [full_name2]
* x500: id_for_first_name , [id_for_second_name] */
```

Your project folder may have all folders and files mentioned in Section 3 Folder Structure. However, please DO NOT include “input”and “output” folders in your final deliverable. You can modify the provided Makefile. Make sure the zip file you are submitting contains all the necessary files and compiles.

Due date for the final submission is Oct. 27, 2021 (Wed.), 11:59pm

7. Grading Policy (tentative)

1. (10%) Interim submission
    a. Correct report content
    b. Completeness of minimum implementation requirements
2. (5%) Appropriate code style and comments
3. (5%) Conformance check
    a. Correct README content
    b. Folder structure and executable names
    c. Print Format
4. (60%) Test Cases
    a. ip1/ : Single File at root level
    b. ip2/: Single Folder
    c. ip3/ : Single Folder and some root level files
    d. ip4/ : 7 folders and some root level files
    e. ip5/ : Symlink
    f. ip6/ : Multi level folders
5. (20%) Code
    a. Correct use of process-related functions such as fork(), wait() and exec()
    b. Correct use of pipe, dup
    c. Correct use of File I/O related functions (fopen etc)
    d. Correct use of Directory functions(opendir, readdir etc)
6. (10%) Extra Credit
    a. (5%)Code
    b. (5%) Test case
        i. Ip7/: Root level hard link
        ii. Ip8/: Subdirectory level hard links