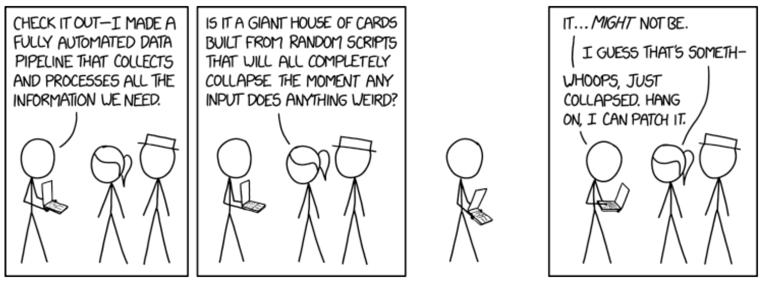
Program Assignment
CPE 357
Systems Programming

Assignment 4


CHECK IT OUT—I MADE A FULLY AUTOMATED DATA PIPELINE THAT COLLECTS AND PROCESSES ALL THE INFORMATION WE NEED.
IS IT A GIANT HOUSE OF CARDS BUILT FROM RANDOM SCRIPTS THAT WILL ALL COMPLETELY COLLAPSE THE MOMENT ANY INPUT DOES ANYTHING WEIRD?
IT... *MIGHT* NOT BE.
I GUESS THAT'S SOMETH-
WHOOPS, JUST COLLAPSED. HANG ON, I CAN PATCH IT.

Write a "findstuff" program.
The findstuff program will let you enter your commands after displaying the prompt findstuff$ (similar to Assignment 3).
Now you should be able to enter some commands and react to them:
Important: I leave the exact formatting and text up to you, but hold up to the premise

find <filename> -s
(Example 1: findstuff$ find project4.c -s) //tries to find the file(s) named "project4.c" in the current directory and all subdirectories.
(Example 2: findstuff$ find project4.c) //tries to find the file(s) named "project4.c" in the current directory

tries to find this specific file in the current directory and all subdirectories (if –s is set).

How to?
Fork a child process which is doing that.
If the search takes longer, you should prompt the user again to be able to enter another search. Hence, you will need several children (we limit it to a max of 10 children). Assign a serial exclusive number to each child, you will need it later for the "kill" command.

Once the search is done, the child interrupts* the read/scanf from the parent process and prints its result.
The result should be something like >**nothing found**< in case nothing was found or if successful, print the file path(s). Print several lines if several files with the same name were found.

-s .. if this flag is set, search in all sub-directories. If not set, search only in the current directory. Also print the search time in the format **hh:mm:ss:ms**

find <"text"> -f:txt -s
(Example 1: findstuff$ find "hello") //searches for the text "hello" in all files in the current folder.
(Example 2: findstuff$ find "blabla" –f:c –s) //searches for the text "blabla" in all c-files and in all subfolders.

The behavior is similar to above, but here, the child tries to find a certain text in all files or if the flag -f is set, only for certain file types (see below). If the flag -s is set, extend the search to all the subdirectories.

-f:XYZ
XYZ representing the file ending. In that case search only in files with this ending.

How to?
You need to open one file at a time, read the content into a malloc'ed memory and try to find the string. List all file path(s) and the time it took for the entire search.

list
(Example: findstuff$ list)

Lists all running child processes and what they try to do. Also displays their serial number. The formating of the output is up to you.

kill <num>
(Example: findstuff$ kill 1) //kills child process number 1 (start counting at 1)

kills a child process, and so ends its finding attemps.

quit or q
(Example: findstuff$ q)

quits the program and all child processes immediatelly.

Have no more than 10 children at a time. Report if the user attempts to exceed that limit and nicely print, that the limit is reached.

*interrupt

Redirect the pipe. Tip: using **read** is easier than scanf.

Learning Objectives

Learning `pipe()`, `dup2()` and how to redirect `stdin`, lots of programing exercise due to string handling, nice use of signals and on top of that a useful program.

How we test

We send a couple of find request on the way on a big HD. So they will need some time (>3sec). Then we check if everything is nicely reporting back.

Submission:

Submit the source code file(s) and the executables: findstuff**.zip**

FAQ

>What happens with the child when its done? E.g. finding the file(s).

1. Report its findings by printing the result.
2. It should end. To avoid a zombie, wait for that specific child with waitpid() (https://linux.die.net/man/2/waitpid) which will be discussed in a separate video. How does the parent know to wait for that child? Signal! And send its PID into a pipe or shared mem!