BSYS - Assignment 1

Implement a *simplified* version of the `find(1)` utility. For preparation purposes, study the system reference manual of the original `find(1)` and perform experiments with this utility.

Name

`myfind` - search for files in a directory hierarchy

Synopsis

`myfind [starting point] [expression]`

`myfind(1)` is a tool that searches for files and directories in a directory hierarchy, beginning from a given *starting point*. Output is determined through an *expression*, which is a chain of filters that are applied from left to right. These filters determine both *which files and folders* are considered and *how output* is formatted.

Note that `myfind(1)` **must** survive the failure of some system calls (e.g. if certain files are not readable by the calling user). Consequently, it is not correct to abort processing altogether when encountering an error.

The options of `myfind(1)` are defined as a proper subset of those defined by `find(1)`. All input valid for `myfind(1)` is likewise valid input for `find(1)`. For all valid input of `myfind(1)`, it should behave identically to `find(1)`.

Options

`<starting point>`

The *starting point* is an optional parameter. It determines the root of a directory tree throughout which `myfind(1)` conducts its search. If not specified, `myfind(1)` uses the **current working directory** as the starting point. In contrast to `find(1)`, `myfind(1)` does not have to support an arbitrary number of starting points.

`-print`

The *`-print` action* prints the full path and file name on standard output, followed by a newline. This is also the default behavior for all files for which the whole expression is true, *unless* `-print` and/or `-ls` are used explicitly in any position of the expression.

`-ls`

The *`ls` action* lists any matching file with a format similar to the one used by `ls -dils`. Specifically, it prints the *inode number*, *number of used blocks*, *permissions*, *number of*

links, *owner*, *group*, *size in bytes*, *last modification time*, and the full path and file name on standard output. The output should be as close as possible to the output of `find(1)`.

```
> find -ls
  3932867      4 drwxr-xr-x   2 alice    users        4096 Nov 16 05:20 .
  3934115    648 -rw-r--r--   1 alice    users      663515 Nov  8  2019 ./2019-11-08
  3932181     32 -rw-r--r--   1 alice    users       30108 May  5  2015 ./2015-05-05
  3934259    516 -rw-r--r--   1 alice    users      528380 Dec  4  2017 ./2017-12-04
  3935724    836 -rw-r--r--   1 alice    users      852211 Nov 14  2021 ./2021-11-14
  3936783    448 -rw-r--r--   1 alice    users      458010 Jun  5  2017 ./2017-06-05
```

`-name <pattern>`

The `-name` test matches the *base file name* (the path with the leading directories removed) against the specified *shell pattern*. The matching is performed with the use of the `fnmatch(3)` library function.

`-type <t>`

The `-type` test matches the *file type* against the specified type `t`. Valid values for `t` are

- `b` for block devices,
- `c` for character devices,
- `d` for directories,
- `p` for named pipes,
- `f` for regular files,
- `l` for symbolic links, and
- `s` for sockets.

`-user <name>|<uid>`

The `-user` test matches the *file's owner* against the specified *user name* or *user ID*.

Step-By-Step

1. Command Line Parsing & Validation

Write a routine that inspects and preprocesses the arguments passed to `myfind(1)`.

- Determine the *starting point*. If the *first argument* is neither an *action* nor a *test*, it must be the name of a file or directory to use as a starting point. If no starting point is specified explicitly, the *current working directory* will be used. Either way, verify that the starting point is a file or directory that exists.
- Parse individual *actions* and *tests*. Every argument other then the starting point must be a defined action or tests.
  - An action (`-print` or `-ls`) does not take additional arguments.
  - A test (`-name`, `-type`, `-user`) is always followed by a single additional argument (*pattern*, *type*, *user name*, or *user ID*).
    - Verify that the *type* is one of the valid single-character values.
    - Verify that the *user name* or *user ID* refer to an *existing user*.
- Make sure you *retain the order* of actions and tests in the *expression*. For each visited file, the expression is evaluated from left to right. Actions and tests can occur in any order and each of them can occur an arbitrary number of times.

Define test cases with different combinations of command line arguments and verify that your implementation is correct.

2. Iterate through the directory tree

Starting from the specified *starting point*, use the library calls `opendir(3)`, `readdir(3)`, and `closedir(3)` to iterate through the directory tree.

- For *starting point* and any visited file, determine whether it is a *directory* or a different type of file. The `stat(2)` system call can be used to determine an entry's file type (e.g. `S_ISDIR(sb.st_mode)`).
- For directories, use `opendir(3)` to open the directory stream, which can be used to iterate through the individual entries in the directory. After processing a directory, don't forget to use `closedir(3)` to close the directory stream.
- Use `readdir(3)` to read individual directory entries from an open directory stream.
- Print the path and file name of every visited file and directory. This is the default behavior of `myfind(3)` for matching files if no explicit *action* is used in the expression.

Define test cases with a variety of files and directories. Verify that `myfind(1)` produces an output similar to `find(1)` called without any tests or actions (e.g. `> find /home/student`).

3. Apply tests and actions for visited files

For each file visited in step 2, apply the individual actions and tests defined in the *expression*. Actions and tests are applied from left to right. If any *test* is encountered that doesn't match the current file, no further tests and actions are applied to the current file and processing continues with the next entry (if any).

- The `-name` test can be conducted using the `fnmatch(3)` library function.
- The `-type` test can be conducted using the output of the `lstat(2)` system call and the macros detailed in the `inode(7)` manual page.
- The `-user` test can be conducted using the output of the `lstat(2)` system call. As the user ID in the output of `lstat(2)` is numeric, *user names* must first be resolved using the `getpwnam(3)` library function.
- The `-ls` action prints information on the current file, must of which is available from the output of the `lstat(2)` system call.
  - Numeric user and group IDs can be resolve to names through the `getpwuid(3)` and `getgrgid(3)` library functions.
  - The `readlink(2)` system call can be used to resolve the value of a *symbolic link*.
  - The `strftime(3)` library function can be used to format the modification timestamp.
- The `-print` action should already have been implemented as part of step 2.

Define test cases with different combinations of expressions that exhibit various actions and tests. Compare the output of `myfind(3)` with the corresponding output of `find(3)`.

Intermediate Checkup

In order to ensure your continuous progress with this assignment, there will be an intermediate checkup during class time. As part of this intermediate checkup, you will demonstrate an intermediate version of your program together with the corresponding source code to your lecturer.