My Find

Introduction

The program my_find aims to implement a search function for UNIX/Linux operating systems without any external libraries (such as the c++17 “filesystem” library). The search can be done recursively and can be case-insensitive. Additionally, the number of sought after files is variable. In the case that the user looks for more than one file at a time, each search is conducted as a new process.

The project was done in C++, with a single file (my_find.cpp), and the process was documented on GitHub.

Usage

./myfind [-R] [-i] searchpath filename1 [filename2] …[filenameN]

Libraries

The following libraries were included for the purposes listed.

Filename and Filepath
#include <string>

Option Parsing
#include <getopt.h>
#include <assert.h>

Process Creation
#include <sys/types.h>
#include <unistd.h>
#include <sys/wait.h>

Directory Iteration
#include <dirent.h>
#include <errno.h>

Standard
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

Search Function

The search function exists as a separate method that has no return value.

As arguments, it takes two essential values: the path the search is to be conducted in, and the name of the file that is sought after. Additionally, it also takes the two boolean flags that signal whether the search should be case-insensitive and/or recursive as input.

Basic Setup

Both a directory pointer, which saves the given directory, and a directory entry pointer, for iterating through the different entries inside the directory, are declared at the start of the method.

```
void search(int pipe[2], bool caseSensitive, bool recursive, std::string path, std::string file
{
   struct dirent *direntp;
   DIR *dirp;
```

To look through the contents of the file/directory, a while loop is used, which will run as long as there are valid files in the file/directory left.

```
while ((direntp = readdir(dirp)) != NULL) //while valid dir is given it iterates through en
{
```

Recursive

The program looks at whether the type of each entry (direntp→d_type) is that of “directory” (DT_DIR) and whether the search function should be called again recursively. If so, the search function is called with the updated path as an argument (path + “/” + direntp→d_type).

```
if(direntp->d_type == DT_DIR) //checks if curr entry is dir
{
   if(recursive)
   {
      search(pipe, caseSensitive, recursive, path + "/" + direntp->d_name, filename); //se
   }
   continue;
}
```

If this is not the case, and the search is not recursive, the search continues normally.

Case-insensitive

In the case that the search is not set to case-insensitive, the name of the current entry and that of the filename will be compared in a case-sensitive fashion. Otherwise, if the search is case-sensitive, the function strcmp() will be used.

```
if((!caseSensitive && !strcasecmp(direntp->d_name, filename.c_str()))
|| (caseSensitive && !strcmp(direntp->d_name, filename.c_str()))) //depending on if case se
{
```

Output

The following output format is requested.

<pid>: <filename>: <complete-path-to-found-file>\n

This was achieved through a pipe.
Firstly, the process id (pid) is obtained by calling the getpid() method. For the absolute path, the array “filepath” is passed on to the realpath() method.

The output is then saved in a string, containing all three necessary elements converted to the corresponding format. Lastly, it is written to the pipe.

```
char filepath[pathconf(".",_PC_PATH_MAX)];
realpath((path + "/" + direntp->d_name).c_str(), filepath); //to get absolute path
std::string output = std::to_string(getpid()) + ": " + filename + ": " + filepath;
write(pipe[1], output.c_str(), output.length());
```

The pipe is closed in the main function. Afterwards, a buffer is declared and initialized, which receives the information from the pipe (fd[0]) until the read() method returns 0, indicating that the pipe has been closed.

```
char buffer[PIPE_BUF];
memset(buffer, 0, sizeof(buffer));
close(fd[1]);
while(read(fd[0], buffer, PIPE_BUF) != 0)
{
   fprintf(stdout, "%s\n", buffer);
   fflush(stdout);
}
```

Closing the Directory Stream

The directory is closed with the help of the closedir(dirp) method, attempted repeatedly in the case that the function is interrupted by a signal from the user.

```
while ((closedir(dirp) == -1) && (errno == EINTR))
   ;
```

Option Parsing

The switch case for option parsing checks for the presence of two options, -i for case-insensitive and/or -R for recursive search, and sets the boolean variables to true accordingly.

The getopt() method inside the switch case scans and reads the input into an integer value (int c).

In the case that an unexpected option is entered (case ‘?’), an error is displayed.

```
int c;
while ((c = getopt(argc, argv, "iR")) != EOF)
{
     switch (c)
     {
      case '?':
         fprintf(stderr, "unknown opt\n");
         break;
      case 'i':
         fprintf(stderr, "Case insensitive requested\n");
         caseSensitive = false;
         break;
      case 'R':
         fprintf(stderr, "Recursive search requested\n");
         recursive = true;
         break;
      default:
         fprintf(stderr, "Default\n");
         return 0;
     }
}
```

Forking

A new child process is created for each file listed in the input. For this, the fork() method is used.

To account for the possibility of multiple files being sought after, the forking happens inside a for loop, which runs the code inside as many times as there are file names in the input.

The id of the process after the fork is saved inside a variable of type pid_d each time. A switch case then determines if the current process is the child (case 0) or the parent, and executes code accordingly. The case -1 handles any errors produced while forking.

Zombie processes are prevented by creating a loop that waits for the termination of child processes.

```c
pid_t childpid; //no zombie processes
while((childpid = waitpid(-1, NULL, WNOHANG)))
{
   if((childpid == -1) && (errno != EINTR))
   {
      break;
   }
}
```

Error Handling

- At the start of the main function
  If the number of arguments, without counting any options (- optind) nor the name of the command (represented by the + 1)

```
if( 1 > argc - (optind+ 1))
{
   fprintf(stderr, "No file and/or path given\nUsage: ./m
   return -1;
```

- In the search function
  If the directory cannot be opened at the given path.

```
if ((dirp = opendir(path.c_str())) == NULL)
{
   perror("failed to open directory");
   return;
```

- In the main function
  Error creating pipe

```
int fd[2];
if (pipe(fd) < 0)
{
   perror("pipe");
   exit(EXIT_FAILURE);
}
```