School of Computing Sciences

Module: CMP-5013A Architectures and Operating Systems
Assignment: OS3 - Unix File I/O in C

Set by: Gavin Cawley gcc@uea.ac.uk
Date set: 25th November 2020
Value: 10%

Date due: 7th December 2020
Returned by: N/A
Submission: Bench inspection

Learning outcomes

This coursework provides further experience of low-level systems programming in C, and the use of POSIX system calls, including error handling. This coursework focuses on file I/O, including accessing the information stored in the i-nodes.

Specification

Overview

You are required to implement a C program, called `executables` that will list, on the standard output stream, all of the executable files (regular files for which the execute permission is set for the user) that are in directories specified using the command line arguments.

You are strongly advised to watch the pre-recorded lectures "*OS2 - System Calls, Processes & Threads*" (slides) and "*OS5 - File System Management*" (slides) that provide the background material for this coursework. The accompanying set of lab exercises "*OS1 - System Calls, Processes & Threads*" and "*OS3 - Unix File I/O in C*" provide useful information on the system calls required. You are also strongly advised to seek help from Dr Bostrom and the teaching assistants during the lab sessions for this coursework on Monday 16th November.

Description

You are required to write a program in the C programming language, called `executables`, that can be used to find the executable programs that are available to the user. This will require two key aspects of the POSIX standard (i) enumerating the files contained in a directory, and (ii) examining the metadata stored in the i-nodes for each file to discover if they are regular files (rather than directories or special device files) and to discover whether they are executable (whether the user has execute permission for the file). The associated labsheet contains some information that will be useful for this task, but it will also require you to search the API documentation (perhaps using the `man` command).

The program should accept command line arguments to specify a sequence of directories to search, e.g.

```
[gcc@cmp-pi42 ~] executable ~/bin /bin /usr/local/bin
```

Alternatively, the command line arguments could provide a colon-separated list of directories. On Unix, there is an environment variable called `$PATH` that lists the directories that are to be searched to find commands executed by the shell. This can be displayed on the console using `echo`:

```
[gcc@cmp-pi42 ~] echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/bin:/usr/bin/site_perl:/usr/bin/
vendor_perl:/usr/bin/core_perl
```

All of the executable programs available to the user could then be listed by entering the command:

```
[gcc@cmp-pi42 ~] executables $PATH
```

If no command line arguments are provided, the program should list the executables in the current working directory.

Suggested order of implementation. Start by writing a program that will list all of the files and directories in a directory for which the path is hard-coded into the program. Next extend the program to investigate the i-node for each file, using `stat`, to determine whether it is a regular file and whether the user has the execute permission bit set, and display only those files. Next adapt the program to accept directory names given as command line arguments (and use the current working directory if none are specified). Lastly extend the program further so that the command line arguments can be colon-separated lists of directories.

Relationship to formative assessment

This coursework benefits from feedback provided on C programming skills provided by CMP-5015A Programming 2 (or comparable prerequisite modules).

Deliverables

During the in-lab bench demonstration you will demonstrate that your programs function correctly (this may include directions from the marker to use particular command line arguments). You will also show your code to the marker and describe its operation and answer any questions the marker may pose.

Resources

- The accompanying pre-recorded lectures "*OS2 - System Calls, Processes & Threads*" (slides) and "*OS5 - File System Management*" (slides) and lab exercises "*OS1 - System Calls, Processes & Threads*" and "*OS3 - Unix File I/O in C*" provide useful background information.
- The `man` command on UNIX-like systems can be used to provide more detailed information on all available POSIX system calls.
- cppreference.com provides an excellent resource for information about C and the standard C libraries.
- The video "*An Introduction to Unix*" provides historical background on the UNIX operating system and an introduction to the Bash shell. The slides are available here, and the associated lab exercises (from week 1) here.

Marking Scheme

Marks will be awarded according to the proportion of the specifications successfully implemented, programming style (indentation, good choice of identifiers, commenting etc.), and appropriate use of programming constructs. It is not sufficient that the program merely generates the correct output. Professional programmers are required to produce maintainable code that is easy for *others* to understand, easy to debug when (rather than "if") bug reports are received and easy to extend. The code needs to be well structured, and written so that it not only solves the problem specified today, but could easily be modified or extended to cater for future developments in the specification without undue cost. The code should be reasonably efficient (for the specified algorithm!). Marks may also be awarded for correct use of more advanced programming constructs or techniques not covered in the lectures. Students are expected to investigate the facilities provided by the POSIX library and the C standard libraries.

- Listing the executable files in a directory (directory may be hard-coded). - 3 marks.
- Listing the executable files in directories specified by the command line arguments. - 2 marks.
- Listing the executable files on the user's search path, using the command `executable $PATH` - 1 mark.
- Quality of implementation (including error handling) - 2 marks.
- Explanation of programs and answers to questions - 2 marks.