School of Computing Sciences

Module: CMP-5013A Architectures and Operating Systems
Assignment: Coursework 2 - Incremental Backup Utility

Set by: Dr Gavin Cawley (g.cawley@uea.ac.uk)
Date set: 26th November 2018
Value: 25%

Date due: *date missing*
Returned by: *date missing*
Submission: Blackboard

Learning outcomes

The aim of this assignment is to gain experience in systems programming, using the C programming language in a Unix environment. On successful completion, the student will have demonstrated the ability to use Unix system calls (in this case primarily relating to the file system) to construct a non-trivial and useful program. This requires the ability to use Applications Programming Interface (API) documentation (as provided by volumes 2 and 3 of the Unix manual, type e.g. `man stat`) to discover how to perform a particular system call. The student will have demonstrated a competence with the C programming language, including memory allocation and deallocation and the use of structures representing compound data types.

Specification

Overview

Systems programming, unlike applications programming, creates software that provides services to other programs or programs for systems administration, rather than programs to be used directly by the users of a computer system. The C programming language is well suited to systems programming, especially for the UNIX operating system, as it allows the programmer to exploit the characteristics of the underlying hardware, and access to the *system calls* provided by the operating system. An important task of the operating system is file-system management. Keeping backups of the user's files is an important task to mitigate against the cost of file-system failures. Taking a backup of the entire file system, especially on a multi-user system, can be extremely expensive. Therefore it is common to use an incremental backup utility, that archives only those files that have been modified since the last back-up of that file was taken. Implementing a simple incremental backup utility provides a good introduction to systems programming.

Description

The task of implementing the incremental backup utility is split into three tasks, of increasing difficulty, with each task providing the basis for the next. Each part tests different aspects of systems programming. The knowledge required for the first task (in addition to fundamental C

programming skills) can be found in the associated labsheet. The later parts require increasing amounts of research to find information on the necessary systems calls. Learning how to use the API documentation is a key skill in systems programming. See the marking scheme given below.

Task #1

The first task is to write a program (`listfiles.c`) to perform a (recursive) traversal of the directory structure, starting with the current working directory, displaying the details (size, access permissions and time of last modification etc.) of each file and then performing the same operation for each of the sub-directories, and so on. The information for each file must be presented in the same format given by the command `ls -l`, e.g.

```
-rw-r--r-- 1 gcc cmpsuppport     326 Oct 26 15:22 Makefile
```

The program does not have to be able to deal with symbolic links, or device (special) files etc., just regular files and directories. This task demonstrates the use of system calls to navigate through the file system and to obtain information about files and directories. Useful system calls include `closedir, getgrgid, getpwuid, localtime, ntfw, opendir, readdir, stat, strftime` (note you may not necessarily need *all* of these system calls to complete this task). The macros `S_ISBLK, S_ISCHR, S_ISDIR, S_ISLNK` and `S_ISREG` may also be handy. This task focuses on the use of systems calls, error handling and passing arguments by reference (which is used by many systems calls).

Task #2

Write a new program (`backupfles.c`), extending the program from Task #1 to display the details of only those files or directories newer than a time/date specified as a command line argument, either as a file, or a quote delimited string of the form `''YYYY-MM-DD hh:mm:ss''`, where `YYYY` is the year, `MM` is the month of the year, `DD` is the day of the month, `hh` is the hour of the day, `mm` is the minute of the hour and `ss` is the number seconds. If a file is supplied instead, the time of it's last modification is used (allowing the last back-up file to be used to specify the cut-off time for the subsequent incremental backup). If no date is specified, a default time/date of `1970-01-01 00:00:00` is to be used.

This task focusses on the processing of command line arguments, and some additional systems calls. The program must support the following command line options:

`-t {<filename>,<date>}` - used to specify the cut-off time for the incremental backup. The `-t` switch must be followed by the name of a file, or a quote delimited string containing a date in the format given above.

`-h` - program prints a help message explaining the usage of the command and then exits.

The switches may appear in any order. The last item on the command line is the directory from which to start the search for files that need to be backed up. Additional system calls that may be useful include `difftime, mktime` and `strptime`.

Task #3

Extend the program from the previous task, once more, to write a program named `backup.c` that implements a simple incremental back-up utility. The program must have one additional switch:

`-f <filename>` - used to specify the name of the archive file to be created or used to restore files from the backup.

Create a symbolic link to the executable called `restore`. If the program is executed using the command `backup`, the program should create an archive file containing all of the files newer than the specified cut-off time/date. If the program is executed via the link, `restore`, it should extract all files and directories into the current working directory. Beware: make sure you always run this program in a safe test directory to make sure you don't overwrite an important file. Also make sure you regularly back your work up on your regular UEA home directory - just in case! When restoring a file, the program should recreate the original access permissions and modification time.

To write to and read from the archive file, you may use the standard C library file I/O routines (`fclose, fopen, fread, fseek, ftell, fwrite` etc.) or the corresponding system calls (`close,creat,lseek,open,read,write` etc.). It is important to think carefully about the format of the archive file before you start this task, as there are a variety of different methods that could be used. The following additional system calls may also be useful: `chmod` and `utime`.

Relationship to formative assessment

The first formative coursework on CMP-5015Y (Programming 2) provides formative feedback on the C programming skills required for this assignment, including pointers, memory allocation and deallocation, the use of structures to implement compound data types, linked lists and basic file I/O. Formative feedback was also available from the teaching assistants for the CMP-5014A labsheet covering UNIX system calls providing access to the file system.

Deliverables

Your solution must be formatted using the PASS program, available on all laboratory machines, to produce a PDF file containing the source code of the three programs. The PDF must then be submitted via BlackBoard. Make sure you are satisfied with the way in which PASS has formatted your code; if there are problems, contact me (g.cawley@uea.ac.uk), however there is only a limited amount of help that I can give at the last moment.

This is an exercise that *can* be done in pairs. If you choose to work in a pair you *must* notify me about your pair members by email by 5pm on Wed 5th December. If you don't, you must complete your work individually. When formatting the submission using PASS, the blackboard IDs and registration numbers of *both* students must be given, and both members of the group must then submit the same .pdf file via Blackboard. Late submissions need to follow the appropriate late hand-in procedure. You can find out this information from the Hub. If you have medical or other problems you can seek extensions to coursework deadlines. However, it is essential you obtain proper documentation in such cases (i.e. a medical certificate). If one member of the pair is granted an extension, the other member of the pair also receives an extension. However, remember that this will apply only to those pairs who notified me that they will work in pairs, as explained above. If you choose to work in a pair, both members of the pair are equally responsible for the joint work. In case of any breakdown of co-operation, you should complete your work individually and notify me by email. You will not receive any extra marks in such a case or for choosing to work individually.

Resources

- `https://stackoverflow.com/` - An excellent site for finding information about specific issues relating to various programming languages, including C. It is important however not to become too reliant on sites such as `StackOverflow`, which are great for details, but don't give the "big picture", so it is difficult to get a good understanding of programming in this way. Note that if you re-use or modify code found on-line, then you must provide a comment giving the URL and a brief explanation of what modifications have been made. Re-using code found on-line without proper attribution would constitute a clear case of plagiarism.

- `https://en.cppreference.com/w/c/language` - A reference website for the C++ programming language, but which also has a section for C. I find it useful for looking up details on C standard library functions.

- `https://www.gnu.org/software/gnu-c-manual/gnu-c-manual.html` - The GNU C reference manual (`gcc` is the GNU compiler collection, which is the C compiler used by Netbeans). This is probably rather more information than you really want!

If you become stuck, please do feel free to discuss problems with the teaching assistants during the scheduled laboratory sessions, or contact me via email (g.cawley@uea.ac.uk) for assistance outside lab hours. I will be running a programming help-desk on Fridays in BIO 0.12, approximate hours 11:30–13:30, 14:30–15:30, 16:00–17:00.

Marking Scheme

Marks will be awarded according to the proportion of specifications successfully implemented, programming style (indentation, good choice of identifiers, commenting etc.), and appropriate use of programming constructs. Make sure no line of code is more than 80 characters long, so that it is displayed correctly on any terminal or printer without truncation. It is not sufficient that the program generates the correct output, professional programmers are required to produce maintainable code that is easy to understand, easy to debug when (rather than if) bug reports are received and easy to extend. The code needs to be modular, where each module has a well-defined interface to the rest of the program. The function of modules should be made as generic as possible, so that it not only solves the problem specified today, but can easily be modified or extended to implement future requirements without undue cost. The code should also be reasonably efficient. As this is a C program, good memory management is required. Marks may also be awarded for correct use of more advanced programming constructs and/or system calls not covered in the lectures.

The incremental marking scheme, which awards marks for specific stages in implementing the full program, is as follows (out of 25 marks):

- Program to perform a recursive traversal of the directory structure, starting with the current working directory. [8 marks]

- Program to display the details of only those files or directories newer than a date specified as a command line argument. [5 marks]

- Incremental backup program that creates an archive containing all of the files that have been created or modified since the last backup was taken, and which can restore files from the backup. [12 marks]