RMIT UNIVERSITY

COSC 1114 Operating Systems Principles
Semester 2, 2021
Programming Assignment 2

Assessment Type	Individual assignment. Marks are awarded for meeting requirements as closely as possible. Clarifications/updates may be made via announcements/relevant discussion forums.
Submission Policy	Submit online to Canvas → Assignments → PrgAsg2 . 2 submissions are required . The first in week 13, and the final submission in start of week 14. Late final submissions are penalized at the rate of 10% per day, pro-rata to 1%. In other words, the formula is floor(hoursLate*10/24) as percentage of the mark received. So 12 hours late for a submission given the mark 20/30 will be penalized as per newMark = origMark * (1 – (floor(hoursLate*10/24) / 100)) = 20 * (1 – 0.05) = 19 to 19/30. NB. In any dispute over late submissions, GitHub commit dates (which can be falsified) will NOT be considered acceptable as evidence.
Submission Due Date	Start of week 14.
Marks	30 (30% of semester)

1. Overview

One of the other large areas in operating systems beyond process management is the domain of File Systems. As you can see in the Group Project title list, there are a lot of them. But they all have some features in common, and in this assignment, we will cover some of these common features.

In this assignment, you will build a very simple file system (VSFS), which is essentially a large text file. You will write a program or script that will insert entries from outside into this FS and vice-versa, as well as many other common file actions. For documentation, you will reflect how certain additions, such as file blocking, FS compression, FS defragmentation, could be implemented into your code (without doing it). For higher marks, you will implement some of these

2. Learning Outcomes

This assessment relates to all the learning outcomes of the course which are:

- Summarise the full range of considerations in the design of file systems, summarise techniques for achieving synchronisation in an operation system,

3. Project Details

3.1 Command Line

You must create a program, called, say, VSFS, that creates a file system called, say, FS, that can be called from the command line and will work as follows. (General format: "VSFS ***command*** FS rags…", where command is {list copyin; copyout; mldir; rm; rmdir…})

1. **VSFS *list* FS** **List the contents of FS in "ls -lR" format**
   a. Format:
   `AAAAAAA NNN OOOOOOOO GGGGGGGG SSSSSSSS DATETIME [PATH/]FILENAME`
   Where
   All elements are separated by a single space char.
   A = attribute. Use the same file attribute as the FS, except where d="-" if the entry is a file, and "d" if a directory.

N = number of hard link if a file or number of subdirs if a fir.
Use “ 1” (3 chars) for file as links not allowed, else use number of dirs. 1 level below.
O,G = owner and group of the VSFS file Use the owner/group of the FS
S = size of the file in bytes
(optional, make size = nyuumber of lines in the file)
Datetime = datetime of the FS file in “ls -l” format.
Path/file = the note name

Example:

```
drwxr-xr-x+ 3 W8431514+ronvs W8431514+None 0 Oct 22  2020 usr/libexec/mc/extfs.d/
drwxr-xr-x+ 1 W8431514+ronvs W8431514+None 0 Oct 12 12:45 usr/libexec/mc/extfs.d/README
```

As you can see, the owner/group strings can vary in length from the standard.

2. VSFS *copyin* FS EF IF — Copy the external file, EF, into the FS as internal file named IF
3. VSFS *copyout* FS IF EF — Copy the internal file IF within FS to external file EF
4. VSFS *mkdir* FS ID — creates empty internal directory ID in FS
5. VSFS *rm* FS IF — Remove internal file IF from FS
6. VSFS *rmdir* FS ID — Remove internal directory ID from FS
7. VSFS *defrag* FS — Defragment FS, removing all deleted entries
8. VSFS *index* FS — (Re-)Create a global index for FS

Where the parameters are defined as follows:

- “IF” can be a path without the leading “/” and ending in a file name but not ending with a “/”.
  - The path referred to here is the internal path, NOT the Unix path.
  - IF should never be “.” Or “..” or “/”
- For *copyin* and *cioyout*, the internal and external paths must already exist else error
- For *mkdir* the ID must NOT already exist else error
- For *rm* and *rmdir*, the IF / ID must/dir exist, else error.
- “EF” can be a filename, or a path ending in a filename. If it is only a file, it is copied to the local directory. If OF is a path, then all folders along the path must already exist. The path is relative to the local directory unless preceded by a “/”
- The command *mkdir* and *rmdir* require a directory, but in this case, the trailing “/” is optional
  - *mkdir* creates a file called “ID”
  - *rmdi*r calls rm for each file within ID and dir containing that path, and then rmdir the ID
- “FS” can be a standard Unix file (and can include a path).
- “ID” is the same as “IF” except that it must be a directory.

Think of the Unix program “zip” and its command line arguments. The ZIP file is the system, and you use “zip” and “unzip” to manipulate the FS, but there are also programs like “WinZIP” that do both.

Suggestion for developers: Get all the above working for files first, then later put in the path handling.

3.2 File format

In the list above, “VSFS” is the name of a Very Simple File System (VSFS) file. For this assignment, we will implement a notes file format, so the VSFS format will have a default extension of “.notes”.

The notes file format is described as follows.

1. It is a text file with variable length records.
2. A record is terminated with “\n”, so a C function fgets() would return it as a string.
3. Maximum record length is 255 characters including “\n”. When using *copyin*, all longer records are truncated.
4. All records starting with “#” are ignored.
5. The first record in the VSFS file starts with “NOTES V1.0”. A valid notes file must contain this text in the first line.
6. A file name record starts with “@” as the first character followed by a “path/…/file” entry. If path is omitted, path = “”. That is, there is no leading “/” in the name. So a plain file “abc” is encoded as “@abc”
7. A directory record is a file record starting with “=” and ending with a path ending in “/”.
8. A content record contains a “ ” (space) in the first character and must succeed a file record. All subsequent content records go in the same file entry, until a non-content record is encountered.
9. A file may be deleted. In that case, the file record and any following content records will have the first character set to “#”.
10. A new file using *copyin* is always appended to the VSFS.

11. If *copyin* is used to replace a file, that file is first deleted, and the new file appended.
12. When using *copyin* to insert a file, always prepend the content records with a " " (space).
13. When using *copyout* to extract a file, always remove the first character (" ") before saving.
14. Only records starting with one of the first characters described above are allowed in the VSFS file. Any other character should result in an error.
15. No two notes should have identical names at the same directory level.
16. All errors should result in a "Invalid VSFS" message to stderr, and exit code of 1 (eg. "exit(1)" (
17. When reading a FS, any deleted records should not be acted upon, but should also not be removed. This way deleted files should remain in the VSFS until explicitly removed.

Sample file:

```
NOTES V1.0
@note1
 This is the first note.  Notice the leading space.
 Another line
@note2
 The next note is empty.
@emptynote
=dir1/
=dir2/
@dir1/note1
 Although this is also note1, it is not in the root dir as the first was.
 Dir2 above is an empty dir
#deletednote
#these lines are ignored when reading in VSFS
@dir3/abc
 This is an error, since dir3 was not created using a directory record.
 This should terminate with an appropriate error message to stderr, and
                error code as per section3.4.
@dir1/note1
 Another error,,,
                Exit code as per 3.4 since note1 already exists in dir1
```

3.3 Midnight Commander, mc.

The program mc ("Midnight Commander") is a program that resembles Windows Explorer, or Finder on the Mac, except that it runs in text mode. The program is already installed on Titan, so just run it using "mc". The program provides a wealth of capabilities for dealing with files and is commonly used to maintain broken Linux systems that are running in single-user mode with mc as a TUI in console mode.

It opens with two screens, which can move independently through the Unix file system. Normally you press TAB to swap screens. You can press function keys such as F5 to copy the highlighted file to the directory in the other screen. This is somewhat equivalent to a drag-and-drop operation.

mc has an interface that allows it to decompress ZIP, TAR.GZ and other compressed files, and render them just as it does on normal files. You simply cursor to a file (say, abc.zip) that contains folder A and B, with files a/aa and b/bb, then if you press return while on abc.zip, it will open up the window with two directories, a, b, and if you enter a, you will find aa. This functionality is called an external file system, or extfs, and mc allows new entries to be added.

You can find all the configured file locations using the command "mc -F", On titan, the extfs folder is "/usr/libexec/mc/extfs.d". If you look at the files here, you will find script files that implement the same interface as the VSFS program described above – even a README that explains the interface. You might like to look at UZIP which uses zip and unzip to maintain zip files, and is written in Perl. Or look at UZOO, which uses the zoo archiver, which has a similar interface to zip, but is written in BASH shell.. 7z, vz, gz, and rzr are also there. Even GitHub is there!

The ultimate goal of this assignment, is for you to create a new script, link it into the mc system, and enjoy a new file system that you built yourself.

While debugging, VSFS can be called from the command line as per section 3.1. But you can also deploy it as an extfs file system within, mc.. This would require the insertion of several settings within your personal mc directories. See the help (F1) for details.. You can add your own extfs file system into the directory **~/.local/mc/extfs.d/.** You nay need to create it,

You run it within mc in one of three ways:

1. Type **"`cd abc,notes/VSFS://`"** on the command line within mc, to explicitly execute this extfs handler.
2. You press alt-C to get a cd prompt, and enter **"cd abc,notes/VSFS://"** into it.
3. You associate the .notes extension with the file system. See f1-Help, then contents, then ext. If you choose this option, then you can just press ENTER when cursored on the file,,

In all cases, you should see one of the panels open up on the notes file system. You can then use F3…F8 to view, edit, copy, move, mkdir, and rm/rmdir entries. Whenever you do any of these, mc will run "VSFS list" to refresh the file list on the panel.

3.4 Error / Exit Handling

If the program works, it must output an exit code of 0. If it errors for any reason, it must print a one-line error message to stderr, and then terminate with a corresponding error/exit code. Wherever possible, use one of the standard Linux exit codes. If your error does not match one of these, then use a non-standard value. In either case, document all possible error codes.

3.5 Submission levels.

This is a rough measure and assumes that all the non-programming parts get full marks.

PA level: Performa all the above activity in sections 3.1–3.4. "mc" compliance is not required.

CR Level: PA level + add the "VSFS "*defrag*" FS" command, which build a new FS, removed deleted entries and sorts the entries into file tree sequence (like ls -lR).

DI level: CR level + automatically create all intermediate directories as needed. (In PA/CR intermediate these must already exist) and error if not. In DI level, they are automatically created.

HD level: DI level +

1. Use the titan program "base64" or similar functionality within your language to encode a binary file into usable text content, then convert *copyin* and *copyout* to encode and decode the files on input to, or output from the file system. So now the FS can also store binary files, and it is still editable with your text editor. Be sure to maintain the leading space in the FS file. favorite
2. Make it mc compliant. Make any and all changes necessary to ensure that mc will use your code Document what you needed to do.

HD+ level: Top level. If the notes file ends in .gz, then use gzip/gunzip to compress the entire FS. One way to do this in C/C++ is to use system, or popen() the programs "gzip"/"hunzip" and read/write from its stdin/stdout pipes using popen(). In other languages, there are similar methods to pipe command output.

BONUS Marks: (This can be a challenge – only attempt if you have time and interest) Include full documentation.

Alternative 1

Download some classical books from https://gutenberg.org/ The book files are pure text. Converting them to notes can be difficult – be careful about the time spent.
Create a program/script to convert them to notes format, with chapter and possibly section headings as files. Use your judgement for section titles.

Alternative 2

A variation of the notes file format of your choice of similar complexity to alternative 1, but prior consent is required.

4. Language Agnostic

For the VSFS code, you may use any language that is installed on titan / saturn / jupiter. They include C / C++, Java, PHP, Perl, Ruby, JavaScript, bash, csh, awk, and more. As long as you can run it, it is acceptable.

Note that all script files need

1. To have the executable bit set (chmod uog+z file)
2. Have the path to the script program in the first line as a comment (#!/bin/sh is the most common)

Most languages allow you to read/build the entire FS in memory using things like associative arrays, so even rebuildin the FS is possible.

5. Report and Submission

In addition to the program solution, you are required to write a DOCX/PDF report describing the following.

1. A syntax diagram
    a. for the command line as you implemented it.
    b. for the notes file format
2. A full "man" page for your program instandard man page format. It would be in section 1.
3. The rationale for using the language that you have used for the implementation.
4. What you would change in order to
    a. build a table at the top to each note, and fseek() there to quickly get to the file.
    b. use fixed length records, and calculate the fseek() instead of using a table.
5. Another application of this file format, or a small variation of it.
    a. One example is the book format suggested in the bonus marks part.
    b. Another suggestion is a Rubric format where all the parts of the Canvas rubric are files within the FS.
    c. Your turn. Describe a third format / application.

6. Submission

1. Submit a ZIP file containing a complete set of source and test files needed to build your solution, ***and***
2. A separate Report in DOCX/PDF format as per previous section

Students are REQUIRED to make 2 submissions of the ZIP and DOCX files, in week 13,14, with the last being the final submission. Only the final submission will be used for marking, but the others are needed to demonstrate progress.

NB. Also note the submission policy at the top of this document.

6.1 Assessment declaration:

When you submit work electronically, you agree to the assessment declaration:

https://www.rmit.edu.au/students/student-essentials/assessment-and-exams/assessment/assessment-declaration

7. Academic integrity and plagiarism (standard warning)

It is your responsibility to ensure that all files you submit are your own work. We will check your submission against other submissions using automated software to check for plagiarism. You must agree with the RMIT Assessment declaration available here:

Academic integrity is about honest presentation of your academic work. It means acknowledging the work of others while developing your own insights, knowledge and ideas. You should take extreme care that you have:

- Acknowledged words, data, diagrams, models, frameworks and/or ideas of others you have quoted (i.e. directly copied), summarized, paraphrased, discussed or mentioned in your assessment through the appropriate referencing methods,
- Provided a reference list of the publication details so your reader can locate the source if necessary. This includes material taken from Internet sites.

If you do not acknowledge the sources of your material, you may be accused of plagiarism because you have passed off the work and ideas of another person without appropriate referencing, as if they were your own.

RMIT University treats plagiarism as a very serious offence constituting misconduct. Plagiarism covers a variety of inappropriate behaviors, including:

- Failure to properly document a source
- Copyright material from the internet or databases
- Collusion between students

For further information on our policies and procedures, please refer to https://www.rmit.edu.au/students/student-essentials/rights-and-responsibilities/academic-integrity

8. Rough Marking Guidelines

A rough marking guide is described below. A more detail marking rubric will be attached to the assignment spec, and it will be the final guide.

Report,docx (5 marks)
As per report section.

Test Strategy (5 marks)
You must devise a set of test strategies appropriate to your language, to 'convince' the markers that your test strategy is complete would discover and document all bugs and error conditions.

Programming Style (4 marks)
You must follow good programming style - avoid use of "magic numbers", name your constants, variables and functions meaningfully, etc. There must be plenty of internal comments describing why is done piece is done that way.

PA level Implementation (4 marks)

CR level Implementation (3 marks)

DI level Implementation (3 marks)

HD level Implementation (3 marks)

HD+ level Implementation (3 marks)

BONUS Marks – Only awarded if fully implemented. (2 marks)

========
(max 30 marks)

For each of the criteria, there are 4 possible marks:

- *Excellent: the solution you have provided is correct as per assignment requirements.*
- *Good: the solution you have provided is good but there are one or two minor issues with your implementation. Your marker will detail the problems in the provided implementation.*
- *Fair: you gave it a go but it's a long way from the required implementation.*
- *No marks: no implementation provided.*

Note that the published rubric for this assignment will apply, even if there are small differences between that, and the marking described herein.