Assignment 1 — Versioning file system

Due Date — 11AM on 3 March 2014

1 Logistics

- Each of the four assignments can be done alone or in a group of two. No request for a group of three will be allowed. You must work on each assignment with a different partner. One of the four assignments must be done alone.
- I never extend deadlines but I'll give sufficient but overlapping deadlines for all four assignments so you can plan your semester. Your second assignment will be released well before the due date of the first assignment. Starting early always helps.
- Absolutely no sharing of code is allowed even if it is a not so important helper function. Your code will be matched against other submissions and submissions from last year. The minimum penalty for plagiarism is an F in the course.

2 Goal

Understand and implement zero cost snapshots in a copy on write log structured file system.

3 Simplifications

- Block size of the file system is 1K bytes. All reads and writes are done in 1K block units.
- Your file system is not implemented over a block device like normal file systems. It is implemented on top of a file on an existing file system. You will treat this file as your backing store i.e. the block device. So block 20 will be the data in this file from 20x1K to 20x1K+1023.
- Maximum file path should be at least 100 characters. E.g. "/dir/subdir/file" is 16 characters.
- Maximum number of files supported by the file system should be at least 1000.
- You will not implement any cleaning. Deleted blocks are wasted.

4 Commands to write

Your program will be executed like this

```
vfs <vfs_filename>
```

If `<vfs_filename>` does not exist, a blank file system is created and an initial checkpoint is done. The checkpoint block number is printed. At this point a command prompt (`>`) should be shown and following commands should be supported:

- `> dir`

  Display contents of current directory.

- `> cd <dir_name>`

  Change to given directory. Optionally show the current directory name in the command prompt. Optionally implement a `pwd` command to print working directory.

- `> read <filename> <block_no> <outputfile>`

  Read the block `block_no` of file `filename` in your file system and write it to the external file (not stored in your file system) `outputfile`.

- `> write <filename> <block_no> <infile>`

  Write 1K bytes at block `block_no` of `filename` read from the start of external file `infile`. The command should create any directories necessary. No meta-deta (checkpoint blocks etc.) should be written to disk by this command. The in-memory meta-deta changes will go to disk when a `checkpoint` command is given.

- `> checkpoint`

  Checkpoint by writing the updated in-memory meta-deta including the checkpoint block to the log and output the block number of checkpoint block to the user. We can call this number the checkpoint block number, the snapshot number, or the version number. Only meta-data blocks should be written by this command and no data blocks should be written.

- `> switch <version>`

  Switches to an old checkpoint. Assume `version` is an old checkpoint block and trust the user. You do not have to validate that it is indeed a correct checkpoint block i.e. I will *not* test your program by giving it an incorrect checkpoint block. After switching, `read` commands should work correctly. Optionally. `write` commands can also work but not required.

- `> delete <filename>`

  Deletes the file from the file system but does not free any blocks. Blocks are never freed in this file system. If I see the directory, the file should not be there. But, if I `exit` and start again, the file *should* be there unless a `checkpoint` was also done. In that case, the file should be permanently gone from all versions *after* that checkpoint but would still exist in old snapshots.

- `> copyfs <version> <outputdir>`

  Copies the given `version` of the whole file system to `outputdir` on an extenal file system.

- `> script <scriptfile>`

  Runs all the commands (one per line) in `scriptfile`. This is a convenience method for testing the file system. You must implement it.

- `> exit`

  Optionally get confirmation if the last command was not a checkpoint.

5 Consistency

If you ever output a checkpoint block number, then that version should *never* get inconsistent in the future and should contain all changes up to that point. This should work even if any number of blocks added after the checkpoint was done are removed or corrupted.

6 Details

You are free in choosing the language (Java, C#, C, C++, Python, etc.) to implement the assignment. You can even go for a FUSE-based file system. In that case, you would need to provide `checkpoint` and `switch` commands while the rest would be handled through FUSE. You are free to choose a design i.e. FAT, i-nodes, etc. that fulfills the requirements. You are free to choose a directory structures (single list of all files with complete paths, or separate list for every directory, or a hash-table etc.) that fulfills the requirements. You don't need free-block management.

Good Luck!