THE UNIVERSITY OF AUCKLAND
NEW ZEALAND

COMPSCI 340 S2 C 2013
SOFTENG 370 S2 C 2013
Operating Systems
Assignment 2 – File system implementation

Due Monday 30th of September 9:30 pm

Introduction

In order to store files on a drive, the drive must have a number of data structures stored on it. These data structures are sometimes referred to as metadata - data about data. This information has to be stored on the drive so that when the drive is attached to a computer the operating system can make sense of what is stored.

What you have to do

You have to implement a file system which provides the ability to store and retrieve data on a virtual drive.

The virtual drive is represented by an ordinary file on your computer. The code to implement the virtual drive is provided in the `drive.py` file. You make a drive by calling the `Drive.format` method. This creates the file on the real machine which represents your virtual drive. All of the data on the virtual disk should be stored as readable ASCII so that by opening the real machine file you can easily see the contents of your virtual drive. The `write_block` method of the `Drive` class writes directly to the underlying real file. Similarly the `read_block` method reads from the real file. Your file system must do all of its work by writing entire blocks to the drive and reading blocks from the drive using only these functions.

The virtual drive can be any number of blocks from 3 to 999 and the block size is 64 bytes.

The file system, even though simple, is quite flexible. There are no limits (up to allowable disk space) on the length of volume and file names, and on the size of the (root) directory and files.

Because all data to be written to the drive must be printable ASCII characters you should always use byte strings like `b'I am a byte string'` as data to be written. The “b” before the string means it is an array of bytes rather than a Python 3 string which could have one character represented by more than one byte, depending on the strings encoding.

There are several classes you have to implement and design in order to get your file system working: a volume class, a file class, and probably a directory class with directory entries.

A Volume

A drive is represented to the operating system as a volume. Formatting a volume means putting extra information on the drive so that it can be recognised and used by the operating system. Most of the volume information has already been decided. Refer to the test file `filesystemtest.py` and the class documentation embedded in the `filesystem.py` file. You may include extra information in your implementation.

The volume information is always stored contiguously in the first blocks of the disk.

e.g. The first block of the `driveA.txt` generated by the `test_new_volume` method should look like this (remember that there are newlines ‘`\n`’ in the file to make it easier to read):

```
1
new volume test
10
x--------x
9

**  0 **
```

The `1` represents the number of blocks occupied by the volume information.

Next is the name of the volume `new volume test`, followed by the number of blocks in the volume `10`.

The bitmap shows that blocks 0 and 9 are in use. Block 0 is actually this block itself and block 9 is being used by the root directory.

The `9` shows the block number where the root directory information starts. Then there are 30 unused bytes (spaces).

The "`**    0 **`" is not actually part of the block it is the separator between blocks, including the block number of the block it follows.

A Directory

This assignment only requires a flat file system. That is, all files are stored in one directory. This directory does not have a limit on the number of files it can contain. You are free to design your directory information in any way you like, but remember that the directory may need to grow beyond the initial block allocated to it.

A File

Files can have names of any length. The filename may not include "/" or newline "\n" characters. When a file is created it has a size of zero. Data can be written anywhere in a file from byte position 0 onwards. If you write data at a position that is currently outside the bounds of a file all of the space up to the position of the new data is then regarded as being allocated to the file. You may do this any way you like, including using a sparse implementation if you wish. Data cannot be read from anywhere outside the bounds of the file.

N.B.

The test file `filesystemtest.py` is useful in helping you work on your solution but you need to program to the specifications above (and in the `filesystem.py` file). The markers will use the original `filesystemtest.py` test file plus another test file which tests things like really long volume and file names. You should be able to run the test files on any machine with Python 3 installed (even a Windows one).

How to begin

I strongly recommend you begin by reading (and understanding) the `Drive` class. In particular, this class shows how you can create a `Drive` object using two different techniques (the `format` and the `reconnect` methods).

Then design the data structures for your file system. You need to produce a drawing showing your data structures and how they relate to each other and to the drive as part of the assignment submission.

As you work on your solution keep running the `filesystemtest.py` test file. The markers will be using this (and another test file) to check your submission. They will also inspect the "drives" (which will be normal text files) produced by your solution when the test file runs. You can check my output "drives" on the assignment web page. Your drives don't need to match these exactly because you have freedom in the way you represent the root directory and store files.

Mark allocation (total 30)

2 marks for each successful test (there are 6 tests in each of two test files). 24 marks.

4 marks for the picture *and* description of your file system data structures. Make this as understandable as possible.

2 marks for the question.

Question

Would your solution work correctly with multiple threads? If the answer is yes, explain how this is guaranteed. If the answer is no, describe a sequence of events which would cause a problem.

Resources

The lectures on file systems.

http://docs.python.org/3/

Submitting the assignment

Make sure your name and upi is included in every file you submit.

Use the assignment drop box to submit a `png` (Portable Network Graphics) or similar drawing of your file system data structures, it should be called `filesystem.png` (possibly with a different file extension). If you use a format different from `png` make sure it can be opened by double clicking on it from Windows in the lab. Yes, you may submit a Word file with a picture in it.

Submit your `filesystem.py` file. All of the classes for your file system should be defined in this file.

Submit your answer to the question as a single text file, called `a2Answer.txt`.

Any work you submit must be your work and your work alone – see the Departmental policy on cheating http://www.cs.auckland.ac.nz/compsci340s2c/assignments/.