Lab - A Simple File System

Introduction

The goal of this lab is to obtain experience with file systems. A file system is a core component of most operating systems and the implementation of this system will provide experience designing complex systems. File systems, by their nature, incorporate aspects of naming, fault-tolerance, concurrency control, and storage management. In addition, issues of security and networking appear in more ambitious designs.

You are required to build a disk emulation library that will emulate a disk on top of a single file. Its interface includes functions to read, write, and flush disk blocks.

Disk library

A file system must be able to store its data in a persistent manner. The persistent medium in your assignment will be disk. Your task is to design a simple disk library that reads and writes 4-Kbyte disk blocks. Your file system will be built on top of this interface.

Since we do not have a raw disk for each of you, your disk library will use a single (large) Unix file to emulate a disk. Your disk library will divide the file into 4-Kbyte blocks. Blocks will be written using a ``block number.'' You will use this number to compute the offset into the Unix file. For instance, disk block 10 will be at byte offset `10 * 4096 = 40960`.

The disk interface is as follows:

I. `int openDisk(char *filename, int nbytes);`
II. `int readBlock(int disk, int blocknr, void *block);`
III. `int writeBlock(int disk, int blocknr, void *block);`
IV. `void syncDisk();`

The calls return an error if the underlying Unix system calls fail.

`openDisk` opens a file (`filename`) for reading and writing and returns a descriptor used to access the disk. The file size is fixed at `nbytes`. If `filename` does not already exist, `openDisk` creates it.

readBlock reads disk block blocknr from the disk pointed to by disk into a buffer pointed to by block. Must complete all due writes, if this function is called after single or multiple writeBlock function call.

writeBlock writes the data in block to disk block blocknr pointed to by disk.

syncDisk forces all outstanding writes to disk. There are chances that some of the writes are still due.

Your task is to implement this interface.

Forth part is bonus. You are required to complete first three parts.

Instructions/Assumptions:

1. You can use any programming language you want. I would recommend Java.
2. A single file could not be larger than 1024 KB.
3. Make appropriate and logical assumptions of your own. Don't forget to mention them in email.
4. You are required to mail your assignment before Friday June 10, 2016 on waqas-ur-rehman@itu.edu.pk.

Helping Code in Java

Following code can help you understand first three functions mentioned above in assignment. You need to implement forth function yourself.

```
package basic.funtions;

public class Driver {

	public static void main(String[] args) {

		String fileName = "temp" ;

		BasicOperations.OpenFile(fileName) ;

		byte[] buffer = new byte[2] ;

		buffer[0] = 100 ;
		buffer[1] = 101 ;

		BasicOperations.writeFile(buffer, buffer.length, fileName, 10) ;


		buffer[0] = 111 ;
		buffer[1] = 110 ;

		BasicOperations.writeFile(buffer, buffer.length, fileName, 5) ;

		byte[] buff = BasicOperations.readFile(fileName, 6) ;
		PrintBytes(buff, buff.length) ;
	}

	public static void PrintBytes(byte[] _bytes, int _length) {
		for(int i = 0 ; i < _length ; i++) {
			System.out.println(_bytes[i]) ;
		}
	}

}
```

```java
package basic.funtions;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.RandomAccessFile;

public class BasicOperations {

	static int STD_BLOCK_SIZE = 2 ;

	public BasicOperations() {
	}

	public static boolean OpenFile(String _fileName) {

		String filePathString = System.getProperty("user.dir") ;
		filePathString = filePathString + "\\" + _fileName  ;
		File f = new File(filePathString) ;

		if(f.exists() && !f.isDirectory()) {
			System.out.println("File already exists!") ;
		} else {
			try {
				if(f.createNewFile())
					System.out.println("File is created!");
			} catch (IOException e) {
				// TODO Auto-generated catch block
				e.printStackTrace();
			}
		}
		return true ;
	}

	public static byte[] readFile(String _fileName, int _fileOffset) {

		String filePathString = System.getProperty("user.dir") ;
		filePathString = filePathString + "\\" + _fileName  ;
		File file = new File(filePathString) ;
		byte[] buffer = new byte[STD_BLOCK_SIZE] ;

		RandomAccessFile raf = null;
		try {
			raf = new RandomAccessFile(file, "r");
		} catch (FileNotFoundException e) {
			e.printStackTrace();
		}

		try {
			raf.seek(_fileOffset) ;
			raf.read(buffer) ;
			raf.close() ;
		} catch (IOException e) {
			e.printStackTrace();
		}
```

```
		return buffer;
	}


	public static boolean writeFile(byte[] _buffer, int _length, String _fileName,
int _fileOffset) {

		String filePathString = System.getProperty("user.dir") ;
		filePathString = filePathString + "\\" + _fileName  ;
		File file = new File(filePathString) ;

		RandomAccessFile raf = null;
		try {
			raf = new RandomAccessFile(file, "rw");
		} catch (FileNotFoundException e) {
			e.printStackTrace();
		}

		try {
			raf.seek(_fileOffset) ;
			raf.write(_buffer) ;
			raf.close() ;
		} catch (IOException e) {
			e.printStackTrace();
		}

		return true ;
	}
}
```