Project – Implementation of TRU File System (TFS) (25%)

Introduction

This course has one major project which is divided into 3 stages. Consult your Course Schedule for precise start and due dates for each Stage. Stage 1 and 2 each contributes 7% towards your final grade. Stage 3 contributes 11% for a Project total of 25% of your final grade. Directions for submitting your Project Stages to your Open Learning Faculty Member for grading can be found in the "Assignment and Project Instructions" document located on your Homepage. A description of each Project Stage and submission requirements follows in the section entitled "Implementation Step – Project Stages". A marking criteria can be found at the end of each Project Stage.

Note: Each stage of this project is tied to the one which follows. Be sure you check your Course Schedule and submit the Stage 1 and 2 in a timely manner as your Open Learning Faculty Member will need time to provide you with feedback before you can proceed to the next stage.

1. Why this project?

File system is one of the important portions of an OS that is used by all applications. File systems come in different forms. Simple devices such as digital cameras and MP3 plays have simple file systems that have limited functionality. Other extremes are the file systems that run on networked machines. This project aims to introduce to you a simple file system. By designing and implementing a simple file system, called the TRU file system (TFS), you will obtain much deeper understanding of the file system internals. Because of the significance of a file system, you receive significant insight into the operation of the OS by doing this project.

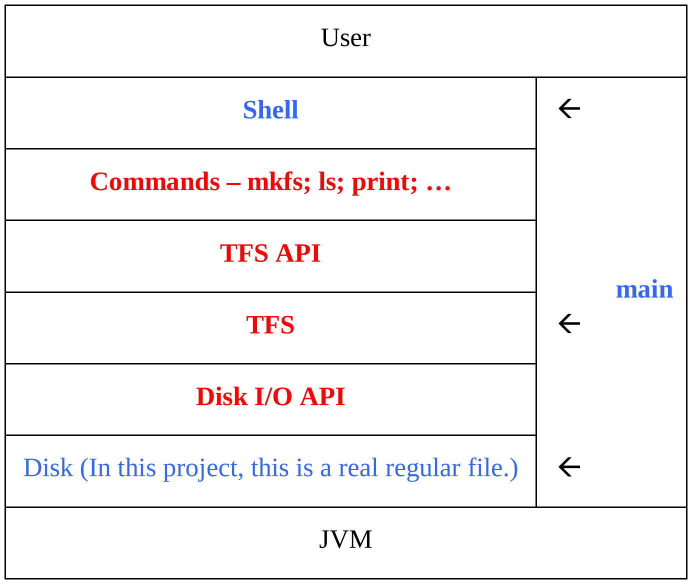
In this project, TFS will be created in one real file in your system.

2. What is required as part of this project?

As part of project, you are expected to design and implement TFS. It is a file system that can be used on a standalone machine. We will make several simplifying assumptions. First, only **a single application is accessing the file system at any given time**. Although this assumption is quite dramatic, it leaves the file system usable in single-tasking embedded system environment such as digital cameras. Secondly, we will implement a simplified interface to the file system with **notable restrictions** such as:

- Limited length filenames – 15 bytes
- A few file attributes – file or directory, file/directory name, size, …
- There is no concept of the current working directory – every file/directory name is given as a full path from the root "/"
- Block size – 128 bytes
- Maximum file size – 65535 bytes
- The disk system will be emulated on a real Java file – **`TFSDiskFile`**

3. Overall diagram


User
Shell
Commands – mkfs; ls; print; …
TFS API
TFS
Disk I/O API
Disk (In this project, this is a real regular file.)
main
JVM

4. Objectives in detail

Shell commands:

You should implement some basic commands that would run from a shell program (**`public class TFSShell extends Thread`**) that will be given. The commands should print proper results and error messages as well.

```
$ mkfs
```

- Make a file system – Make new PCB and FAT in the file system

```
$ mount                                         -
```

- Mount a file system – Copy PCB and FAT in the file system into the main memory

```
$ sync
```

- Synchronize the file system – Copy PCB and FAT in the main memory back to the file system on the disk

$ **`prrfs`**

- Print PCB and FAT in the file system

$ **`prmfs`**

- Print PCB and FAT in the main memory

$ **`mkdir`** `directory`

- Make a directory if it does not exist

$ **`rmdir`** `directory`

- Remove a directory if it is empty

$ **`ls`** `directory`

- List file or directory names in the directory, with size

$ **`create`** `file`

- Create an empty file if it does not exist

$ **`rm`** `file`

- Remove a file

$ **`print`** `file position number`

- Print `number` characters from the `position` in the file `file`

$ **`append`** `file number`

- Append any `number` characters at the end of the file if it exits

$ **`cp`** `source_file destination_directory`

- Copy a file into a directory if they exit and `source_file` does not exist under `destination_directory`

$ **`rename`** `source_file destination_file`

- Rename a file `if source_file` exists and `destination_file` does not exit

$ **`exit`**

- Exit from the shell, i.e., shutdown the system

TFS APIs:

Your TFS should implement the following application programming interface (API) in **`public class TFSFileSystem`**. The methods should return appropriate return values, including errors (negative numbers).

`public static int` **`tfs_mkfs`**`()`

- Create the file system on the hard disk

`public static int` **`tfs_mount`**`()`

- Mount the file system

`public static int` **`tfs_umount`**`()`

- Unmount the file system

`public static int` **`tfs_sync`**`()`

- Synchronize the file system

`public static String` **`tfs_prrfs`**`()`

- Print PCB and FAT in the file system

`public static String` **`tfs_prmfs`**`()`

- Print PCB and FAT in the main memory

`public static String` **`tfs_exit`**`()`

- Unmount the file system, and call tfs_dio_close()

`public static int` **`tfs_open`**`(byte[] name, int nlength)`

- Open the given file

`public static int` **`tfs_read`**`(int file_id, byte[] buf, int blength)`

- Read bytes from disk into `buf`

`public static int` **`tfs_write`**`(int file_id, byte[] buf, int blength)`

- Write bytes from `buf` into disk

`public static int` **`tfs_seek`**`(int file_id, int position)`

- Set the new file pointer

`public static void` **`tfs_close`**`(int file_id)`

- Close the given file

`public static int` **`tfs_create`**`(byte[] name, int nlength)`

- Create the given file

`public static int` **`tfs_delete`**`(byte[] name, int nlength)`

- Delete the given file

`public static int` **`tfs_create_dir`**`(byte[] name, int nlength)`

- Create a directory

`public static int` **`tfs_delete_dir`**`(byte[] name, int nlength)`

- Delete a directory

Disk I/O APIs:

You should implement the following API for the emulation of disk input/output in **`public class TFSDiskInputOutput`**. The methods should return appropriate return values, including errors (negative numbers).

`public static int` **`tfs_dio_create`**`(byte[] name, int nlength, int size)`

- Create a disk file of `size` blocks. **The disk file is a real file in your system, in which TFS is implemented.**
- Return 0 if there is no error.

`public static int` **`tfs_dio_open`**`(byte[] name, int nlength)`

- Open a disk file
- Return 0 if there is no error.

`public static int` **`tfs_dio_get_size`**`()`

- Get the total # of blocks of the disk file
- Return 0 if there is no error.

```
public static int tfs_dio_read_block(int block_no, byte[] buf)
```

- Read a block from the disk file
- Return 0 if there is no error.

```
public static int tfs_dio_write_block(int block_no, byte[] buf)
```

- Write a block into the disk file
- Return 0 if there is no error.

```
public static int tfs_dio_close()
```

- Close the disk file
- Return 0 if there is no error.

5. Implementation strategies

- Structure of TFS on a hard disk (being emulated on a real Java file)

BCB	PCB	FAT	Data blocks

  - Directory structure – Linear list; it contains file attributes as well
  - Allocation method – FAT
  - Free space management – Linked list
  - PCB
    - Two pointers to the root directory and the first free block
    - Size of FAT
    - Number of data blocks

- In-memory data structures
  - PCB
  - FAT
  - System-wide **file descriptor table (FDT)**
    - Combination of file descriptor table, open file table and inode table in the lecture slides of the chapter 11
    - Name
    - Directory or not
    - Starting block number
    - File pointer – Offset where the process reads from or writes to
    - Total size in bytes

- `TFSFileSystem`
  - `tfs_mkfs()`
    - Initialize PCB and FAT in disk
    - PCB in disk -> memory
    - FAT in disk -> memory
  - `tfs_mount()`
    - PCB in disk -> memory
    - FAT in disk -> memory
  - `tfs_umount();tfs_sync()`
    - PCB in memory -> disk
    - FAT in memory -> disk
  - `tfs_exit()`
    - `tfs_umount()`
    - `tfs_dio_close()`
  - `tfs_open()`

    - Search the first block number: PCB in memory -> FAT in memory
    - Create a new entry in FDT
  - `tfs_read();tfs_write()`
    - File descriptor -> FDT -> FAT in memory -> Disk
- `tfs_close()`
    - Delete the entry in FDT

Implementation Steps

Project: Stage 1 (7%)

Instructions

- To begin Stage 1, go to your Home Page and locate the “Project Stage 1_student Java files” folder which contains the Java files you need for this stage.
- `TFSDiskInputOutput.java`, `TFSFileSystem.java`, and `TFSShell.java` include method definitions.
- **First**, you **should** decide data structures for PCB, FAT and directory. Then, include a detailed explanation about them with your `TFSFileSystem.java`.
- **Second**, you **should** implement the next methods.
  - `TFSDiskInputOutput`
    - All methods, i.e., Disk I/O APIs.
  - `TFSFileSystem`
    - `tfs_mkfs()`
    - `tfs_prrfs()`
    - `tfs_exit()`
  - `TFSShell`
    - `mkfs()`
    - `prrfs()`
    - `exit()`

Report Submission Details

- Java source files
- A detailed explanation about all data structures and methods should be included in Java files.
- Screen shots that show how your programs work.

The following marking criteria will be used to grade your work. The total number of marks is out of 10.

Marking Criteria	**Weighting**
The detailed explanation of data structures and methods which demonstrates sound logic and correct application of course concepts.	/2
No syntax error: All requirements are fully implemented without syntax errors. Submitted screen shots will be reviewed with source code.	/4
Correct implementation: All requirements are correctly implemented and produce correct results Submitted screen shots will be reviewed with source code.	/4
Total	/10

Project: Stage 2 (7%)

Instructions

You can begin Stage 2, once you receive feedback from your Open Learning Faculty Member for Stage 1. If you did not complete Stage 1 correctly, this feedback will include the Java files needed to proceed with Stage 2. If you are confident with your code in Stage 1, then you can use your code for Stage 2. Contact your Open Learning Faculty Member if you have any questions or concerns.

- TFSFileSystem.java includes the code of the next private TFS APIs. These private TFS APIs could be used for the implementation of public TFS APIs.
    - Disk related utilities
        - `int _tfs_read_block(int block_no, byte buf[])`
        - `int _tfs_write_block(int block_no, byte buf[])`
    - FDT related utilities
        - `int _tfs_open_fd(byte name[], int nlength, int first_block_no, int file_size)`: Create a new entry in FDT for a file or directory; return the file descriptor
        - `int _tfs_seek_fd(int fd, int offset)`: Change the file pointer to `offset`
        - `void _tfs_close_fd(int fd)`: Remove the file descriptor from FDT
        - `int` **`_tfs_read_bytes_fd`**`(int fd, byte buf[], int length)`
        - `int` **`_tfs_write_bytes_fd`**`(int fd, byte buf[], int length)`
        - `int` **`_tfs_get_block_no_fd`**`(int fd, int offset)`: Find the block number for the `offset` in the file represented by `fd`
        - `int` **`_tfs_read_directory_fd`**`(int fd, byte[] is_directory,`
        - `byte[] nlength, byte[][] name, int[] first_block_no,`
        - `int[] file_size): byte[][16] name`; Read a directory into the main memory from the file system; It returns a negative number if there is an error, otherwise the number of files and subdirectories under the directory
    - PCB related utilities

    - `void _tfs_write_pcb()`: Write PCB back into the disk
    - `void _tfs_read_pcb()`: Read PCB from the disk into memory
  - FAT related utilities
    - `void _fts_read_fat()`: Read FAT from the disk into memory
    - `int _tfs_get_block_fat()`: Get a free block
    - `void _tfs_return_block_fat(int block_no)`: Return a free block
    - `int _tfs_attach_block_fat(int start_block_no, int new_block_no)`: Attach a block at the end of a file having the start_block_no
  - Block handling utilities
    - `int _tfs_get_int_block(byte[] block, int offset)`: Get an integer from a block
    - `void _tfs_put_int_block(byte[] block, int offset, int data)`: Put an integer into a block
    - `byte _tfs_get_byte_block(byte[] block, int offset)`: Get a byte from a block
    - `void _tfs_put_byte_block(byte[] block, int offset, byte data)`: Put a byte into a block
    - `byte[] _tfs_get_bytes_block(byte[] block, int offset, int length)`: Get bytes from a block
    - `void _tfs_put_bytes_block(byte[] block, int offset, byte[] buf, int length)`: Put bytes into a block

- **First**, you may need to include additional routines to handle in-memory data structures, such as PCB, FAT, **FDT**. The detail explanation about **FDT should** be included in `TFSFileSystem`.
- **Second**, you **should**
  - Create PCB, FAT and FDT in the main memory in `TFSFileSystem()`
  - Create the root directory (initially empty) within `tfs_mkfs()`
- **Third**, you **should** implement the next methods.
  - `TFSFileSystem`
    - `tfs_mount()`

    - `tfs_umount()`
    - `tfs_sync()`
    - `tfs_prmfs()`
  - `TFSShell`
    - `mount()`
    - `umount()`
    - `sync()`
    - `prmfs()`

Report Submission Details

You need to submit a report that consists of:

- Java source files
- Detailed explanation about all data structures and methods should be included in Java files.
- Screen shots that show how your programs work.

The following marking criteria will be used to grade your work. The total number of marks is out of 10.

Marking Criteria	**Weighting**
The detailed explanation of data structures and methods which demonstrates sound logic and correct application of course concepts.	/2
No syntax error: All requirements are fully implemented without syntax errors. Submitted screen shots will be reviewed with source code.	/4
Correct implementation: All requirements are correctly implemented and produce correct results Submitted screen shots will be reviewed with source code.	/4
Total	/10

Project: Stage 3 (11%)

Instructions

You can begin Stage 3, once you receive feedback from your Open Learning Faculty Member for Stage 2. If you did not complete Stage 2 correctly, this feedback will include the Java files needed to proceed with Stage 3. If you are confident with your code in Stage 2, then you can use your code for Stage 3. Contact your Open Learning Faculty Member if you have any questions or concerns.

- We will use the next directory structure. However, you can use your own directory structure if you want.

```
    // the total number of entries
    int noEntries;  // it has meaning only in the first block
    int parentBlockNo;  // the first block number of the
parent dir
    // each entry has
    byte isDirectory;  // 0: subdirectory, 1: file
    byte nLength;  // name length
    byte reserved1;  // reserved
    byte reserved2;  // reserved
    byte[16] name;  // not a full path
    int firstBlockNo;  // the first block number
    int size;  // the size of the file or subdirectory
    // the size of each entry is 4 + 16 + 4 +4 = 28 bytes
    // 128 % 28 = 4 => one block can hold maximum 4 entries
```

- **First**, you **should** update the next method in `TFSFileSystem`.
  - `_tfs_close_fd(int fd)`
    - **Update the entry for the file or directory in the parent directory, if there is a change**
    - Destroy the entry of `fd` from FDT

- **Second**, you **should** implement the next directory related utilities in `TFSFileSystem`.
  - `_tfs_search_dir(byte[] name, int nlength)`
    - Return the first block number of the parent directory in which `name` exists
    - **`name` contains a full path**
  - `_tfs_get_entry_dir(int block_no, byte[] name, byte nlength, byte[] is_directory, int[] fbn, int[] size)`
    - Get the entry for `name` from the directory of which the first block number is `block_no`
    - `name` is not a full path
    - `isDirectory`, `fbn` and `size` are arrays of one element
    - Return a negative number if `name` does not exist in the directory
  - `_tfs_create_entry_dir(int block_no, byte[] name, byte nlength, byte is_directory, int fbn, int size)`
    - Create an entry for `name` in the directory of which the first block number is `block_no`
    - `name` is not a full path
    - The whole size of the directory might be changed
  - `_tfs_delete_entry_dir(int block_no, byte[] name, byte nlength)`
    - Delete the entry for `name` from the directory of which the first block number is `block_no`
    - `name` is not a full path
    - The whole size of the directory might be changed
  - `_tfs_update_entry_dir(int block_no, byte[] name, byte nlength, byte is_directory, int fbn, int size)`
    - Update the entry for `name` in the directory of which the first block number is `block_no`
    - `name` is not a full path
- **Third**, you **should** implement the next TFS APIs.
  - `tfs_open(byte[] name, int nlength)`

        - Return file descriptor for the file or directory
        - **`name` has a full path for the file or directory**
        - Need to search `name` from the root directory
    - `tfs_read(int fd, byte[] buf, int blength)`
        - Read `blength` bytes in `buf` from `fd`
        - Return the number of bytes read
    - `tfs_read_dir(int fd, byte[] is_directory, byte[] nlength, byte[][] name, int[] first_block_no, int[] file_size)`
        - Read all entries in the directory `fd` into arrays
        - Return the number of entries
    - `tfs_write(int fd, byte[] buf, int blength)`
        - Return the number of bytes written into the file or directory
    - `tfs_seek(int fd, int offset)`
        - Return the new file pointer
    - `tfs_close(int fd)`
    - `tfs_create(byte[] name, int nlength)`
        - Create a file
        - `name` contains a full path
    - `tfs_delete(byte[] name, int nlength)`
        - Delete a file
        - `name` contains a full path
    - `tfs_create_dir(byte[] name, int nlength)`
        - Create a directory
        - `name` contains a full path
    - `tfs_delete_dir(byte[] name, int nlength)`
        - Delete a directory
        - `name` contains a full path
- **Fourth**, you **should** implement all other commands in `TFSShell`.

Report Submission Details

You need to submit a report that consists of:

- Java source files
- Detailed explanation about all data structures and methods should be included in Java files.
- Screen shots that show how your programs work.

The following marking criteria will be used to grade your work. The total number of marks is out of 10.

Marking Criteria	**Weighting**
No syntax error: All requirements are fully implemented without syntax errors. Submitted screen shots will be reviewed with source code.	/5
Correct implementation: All requirements are correctly implemented and produce correct results Submitted screen shots will be reviewed with source code.	/5
Total	/10