CS 377 : OS LAB Assignment
File System - Part I

1 Overview

We will call the system you developed in assignments #8 and #9 as the *OS simulator*. In this assignment, you will develop the OS simulator further.

Implement a file system in your OS simulator. It should have the following capabilities:

- Each user should be able to create a directory structure in the form of a tree
- A user should be able to 'walk' his/her directory tree and operate on files contained in the directories
- A user should be able to create new files and perform read and write operations on files
- The file system should perform appropriate disk space allocation to support creation of directories and files.

2 Simulating the Disk

You are expected to implement the disk as a large file in your own account.

1. You should decide the size of a disk block in this virtual disk.
2. You should write your own function to perform a *read* operation on the virtual disk. It should take two parameters—(i) a block# in the disk, and (ii) a data area in the OS simulator where the data read from the disk block should be kept.
3. Similarly, you should write your own function to perform a *write* operation.

3 Features of the File System

1. **Superblock**: The file sytem uses a *superblock*. It is typically the first disk block in the part of the disk used to house the file system. You

may assume it to be disk block 0 in your disk. The superblock has the following fields in it:

(a) Number of disk blocks in the file system

(b) Disk block number of the file system's root directory. (The root directory is assumed to fit into a single disk block.)

(c) Number of blocks used for the *meta-data*, i.e., control data of the file system itself. The meta-data would include a *free list*, which contains disk block numbers of all disk blocks that are currently free.

(d) Any other fields you may require (you should be able to justify why the field must exist in the superblock).

2. **File space allocation:** The file system allocates disk space to a file on a needs basis, i.e., when data to be stored in a file needs more disk space. It uses the *linked allocation* scheme.

3. **Directory hierarchy:** The directory hierarchy of a user is governed by the following spec:

   (a) Each directory can fit into a single disk block. You should use a good structure for the directory; initially a sequential arrangement would do.

   (b) To begin with, the file system's root directory contains *home directories* of all its users.

   (c) To begin with, the *current directory* of a user is the same as her/his home directory.

   (d) The user can change his/her *current directory* through a "change directory" command. The syntax of this command is identical with the corresponding Linux command.

   (e) A process created by a user can access a file in the user's current directory simply by using the file's name. The process can access files that are not in the current directory by using a *path name.* The syntax of writing path names is identical with path names in Linux.

4. **Directory entries:** Design your own format for a directory entry, taking care to provide fields to support all operations on directories and files. (If necessary, refer to the directory entry format done in class for guidance.)

Provide a field to store protection information for the file described by the directory entry.

5. **File Manipulation:** File manipulation in a process is governed by the following spec:

   (a) All files are byte-stream files and are used as sequential files. The file system keeps track of the *current position* of a process in a file.

   (b) A *seek* command changes the current position in the file to the byte indicated in the command. The seek command can be used in an absolute sense or relative to the current position in the file. If the result of a seek command is a byte number that exceeds the current size of the file, the file is appended with blanks upto the byte mentioned in the seek command.

   (c) A read/write command indicates the number of bytes that are to participate in the operation. In the case of a write operation, the command also indicates the bytes to be written into the file.

   (d) A process can have many files open at a time. (You could put a reasonable limit on this number.)

   (e) The file system uses a *file control block* (FCB) for each file opened by a process.

   (f) At a read operation on a file, the file system reads the disk block that contains the data to be read.

   (g) At a write operation on a file, the file system writes the data into the correct disk block taking care to ensure that other data in the file is not affected.

4 Ensuring Consistency of File System Meta-data

When a process performs an operation (read/write/create directory), your file system would access its meta-data (such as the free list). Since many processes are running concurrently, these accesses would be concurrent. To avoid race conditions, you must perform all operations on meta-data under mutual exclusion.

5 Controlling the Simulation

1. File `init`, which controls operation of the main process, contains the following commands **in addition to the commands described in specifications of assignments 7 and 8**:

   (a) `uname <user_name>`
   The file system should check whether the home directory for the named user already exists. If so, it should simply ignore this command; otherwise, it should create the home directory for the user.

   (b) `mkdir <user_name> <directory_name>`
   The file system should make the new directory in the current directory of the user.

   (c) `cd <user_name> <directory_name>`
   Change the current directory of the user.

   (d) The `create` command now takes an additional parameter, which is the name of the user who created the process. The command new syntax is:

   ```
   create <user_name> <process_no> <process_size>
   ```

   where

   `<user_name>` is the name of a user (Do not put '<' and '>')

   `<process_no>` is an integer

   `<process_size>` indicates the size of the logical address space of the process, i.e., the number of pages in the logical address space of the process. Note that pages are numbered from 0 onward.

   `<page_frames_allocated>` is a spec `m-n` where `m` and `n` are integers. Note that a process is permanently allocated a contiguous set of frames of memory starting on page frame `m` and ending on page frame `n`.

2. The file that controls operation of a user process can contain the following commands **in addition to the commands described in specifications of assignments 7 and 8**: (*Note:* Parameters of commands are separated by single blanks.)

   (a) `open <file_name>`

(b) `seek <absolute_byte#>` or
`seek relative <displacement_from_current_position>`

(c) `read <file_name> <#bytes>`

(d) `write <file_name> <#bytes> <value_to_be_written>`

6 Producing a Trace During Operation

Your system must produce a trace in which it writes all information relevant to file operations performed in the system.

For example, for a read command from a file, the trace MUST indicate which disk block is read from. It may also contain other interesting information. For a write command, it should indicate whether a new disk block is allocated to the file (and, if so, its disk block number).