

Some Filesystem Design Notes

[An **Example** of the components Required for the implementation]

1.1 Directory Structure (Define the type of Directory Structure you expect to use)

For example:

File Allocation (FA) Structure: logical or user level directory of files and folders

Describe what the FA structure is for describing the file, what each entry contains, and how it fits into the overall design of the filesystem.

Show the format and detail of entries in the structure/table with enough detail to completely specify the data structures required for implementation. Describe storage sizes required for entries and types of identifiers. Include an example of the content of the directory structure for a set of files/directories.

Differentiate the differences and descriptive details of directory descriptive data structures vs. file descriptive data structures in the logical directory.

Physical Allocation (directory of storage units on the disk and how they are related/allocated to files)

You will need to decide how the disk is to be organized in terms of blocks of bytes that become the 'allocation units' (size of a storage unit). As a file is created and grows it is through allocation units.

Describe storage unit size, how each storage unit on the disk is allocated and associated with the file, directory or system space to which it belongs (we can call them storage containers).

Show an example of allocation of the units to each of the functional storage containers

Describe how you address storage units on the disk.

Describe the integration of the logical and physical directories that are defined above.

Definitions

Include any definitions that describe naming conventions or other terms you use for referencing storage units, files, directories, etc.

2.1 Details of Implementation

Overview

Describe storage unit sizes, overhead for storage, location of various filesystem entities on the virtual disk, etc.

3.1 File Interface Code Definitions and Pseudo-code

Define how your filesystem functions will reference files (for example file handles when open) and the data structures you will design to support the method of referencing the files. [do you need tables of data structures to manage open files, positions in files, etc.?]

Define how folders are referenced.

Define how data are read/written and or buffered between disk and user application.

Define and describe each of the functions that you have implemented to manipulate the filesystem, files and folders. Include the function calls and parameter definitions. Be sure to include documentation of any special features or functionality you are including.

Include a description of the function (and its functionality) with details in English. Also include pseudocode of the function implementation.

For example:

Open File

```
/*  
 * Returns a pointer to a file descriptor that represents the file or directory specified by path  
 * Returns NULL if the file or directory does not exist  
 * <be sure to include a description of the arguments and return values, including types>  
 */  
FileDescriptor *fileOpen(char *path);  
:
```

Create File

```
/*  
 * Creates a new file with the specified directory path and returns a pointer to a handle to that file  
 * returns NULL if the file already exists  
 */
```

Create Directory

```
:  
:
```

Delete File

```
:
```

4.1 Notes on programming language issues and related use of File I/O functions of Memory Mapped Files

You are to implement your system through memory mapped files. You will have to define and construct the functions that perform the actual manipulation of the data on your virtual disk (this includes directory data and file data). You will need to create the equivalent of the functions that the OS system library or the C run-time library provides.

That is, you need to implement the real (physical) file operations for your file system. You are to explain and detail the system or run-time library functions you will be using and how you will use them. Include enough detail to show you understand the functions, their features and limitations. Show prototypes and argument definitions for each function.

1. Example – using the memory mapped files facility

As an alternative to standard file I/O, the kernel provides an interface that allows an application to map a file into memory, meaning that there is a one-to-one correspondence between a memory address and a word in the file. A programmer can then access the file directly through memory, identical to any other chunk of memory-resident data—it is even possible to allow writes to the memory region to transparently map back to the file on disk.

POSIX.1 standardizes the `mmap()` system call for mapping objects into memory.

```
mmap( ):  
void * mmap (void *addr,
```

```
size_t len,  
int prot,  
int flags,  
int fd,  
off_t offset);
```

A call to `mmap()` asks the kernel to map `len` bytes of the object represented by the file descriptor `fd`, starting at `offset` bytes into the file, into memory. If `addr` is included, it indicates a preference to use that starting address in memory.

A typical use is (you need to explore the meaning of the parameters)
`p = mmap (0, len, PROT_READ, MAP_SHARED, fd, 0);`

In your filesystem, the logical directory is an area of disk. Consider the root directory. It is stored in some area of the disk that you design/specify. The directory is effectively an array of data structures, where each array element describes a file. By locating the root directory in a fixed, known location, you can always find the starting point of the directory of your files.

You will need to develop functions to access the root directory in the memory mapped file and traverse the directory, modify entries, add entries etc. as part of your implementation of the Create/ Delete, Open/Close file functions.

The virtual disk is organized in allocation units of blocks. Directories and files are comprised of blocks that store the data. Directory entries are to be organized in blocks as well. The management of blocks allocated to the disk and to individual files is a part of the file system.