CIS 3207 Project 4
Constructing a File System and the Related IOCS[1] Functions

1 Introduction

Operating systems need to provide a uniform, logical view of information storage. The operating system abstracts from the physical properties of its storage devices to define a logical storage unit, **the file**. Files are mapped by the operating system onto physical devices. A file is simply a *named collection of related information* that is recorded on secondary storage.

The information in a file is defined by its creator. A file may have a certain defined structure, which depends on its type, or the type of data stored in the file. The data may also be unstructured, i.e., a sequence of bytes.

A **file system** is comprised of a sequence of bytes allocated as a storage container on a physical storage device. The file system consists of two distinct parts: a collection of files, and a directory structure that organizes and provides information about all of the files in the system. The storage container is comprised of data within the files, and meta data that describes the files (the file system directory). Through the file system directory, a user has access to files, the file descriptions, the location of file content and the file content itself.

This project has been developed to help you understand the concepts of a file system, the relationship between files and the file system, and the requirements and structure of a directory of a file system. In this project, you will design a file system and implement it on a disk within an operating system. A good introduction to file systems is in your Three Easy Pieces textbook, Chapters 39 and 40. Information about mass storage/disk systems and I/O management can be found in Chapters 36 and 37. Additional references are given at the end of this document.

Generally, a file system encompasses all of the storage of a device. That is, we generally consider a storage device, such as a disk, as a contiguous set of bytes, and the bytes are available to the file system. The file system gives structure to the storage elements (bytes) on the device. In fact, there is a *logical view* for the contents of a file, and a *physical view*. One can consider that the file system provides the mapping between these views. In this project, "**your file system**" will be created within a single file (allocated storage of bytes) of the operating system disk on which you are developing the file system. That is, you are creating a 'virtual disk', implemented within a file of the operating system disk. (Remember, a file is just a sequence of bytes; you can impose any data structure(s) on this sequence of bytes. In this project, the sequence of bytes of this file will comprise your entire file system).

For the project, you will allocate a large file on your system disk (one large enough to hold both the directory and all the files to be stored in this file system) that can be **logically viewed** as a storage device composed of contiguous bytes. We can call this file a 'virtual disk'. You will give structure to those bytes. You will create a directory that must include a component that describes the allocation of bytes to files and includes information about the files (name, creation date, size, etc.). The file system must describe the allocation of bytes to files and bytes to the directory system; i.e., all bytes that are within the disk must be accounted for and tracked. The directory will be a "file" or allocated area stored within your storage device (i.e., in your file system). [The

[1] IOCS: I/O Control System

directory of the file system could be a special file within your device or just a defined allocation of space on your virtual disk.]

The file that you use as your virtual disk does not have to be physically contiguous (i.e., bytes adjacent to one another). The underlying operating system (e.g., Linux) takes care of the physical allocation of all files, making them appear to be logically contiguous. You can create a large file on the operating system's disk and ignore the real physical organization of the bytes in the file, viewing the storage as **logically contiguous** (your view of the disk you are creating). You can create the file and write bytes for the length of the fie you need to create.

That is, a starting point for this project is to create a large file on a disk that represents "your disk", or virtual device. Your objectives in this project are to create the file system for that virtual device and develop the input/output control software that provides basic file operations on the device. These operations will include: creating and deleting files, opening, closing, reading and writing files, and directory printing for your file system.

2 Your Tasks

1. Design the overall file system
    a. This file system is to have a directory structure that can store (sub)directories and files.
    b. Files and directories can have names with as few as 8 characters along with a 3 character filename extension (file type descriptor). (this is a minimum requirement; you could have longer filenames)
    c. Files can be up to 32768 characters (bytes) in size [again, a minimum requirement.]
    d. Files should not be contiguous within the virtual file system 'disk'.
    e. The allocation units, or blocks, on the disk you are to create are each 512 bytes.
    f. Directory structure
        i. The directory is to be extensible; i.e., all of the elements in the directory can be linked (this does not imply by a linked list) so that additional file/directory entries can be added, or entries removed when files are deleted.
        ii. Every entry in a directory is to have a last modification date/time stamp.
    g. You are to select and design the specific details of the layout of a directory, including the root directory. Your design should include:
        i. Structure and organization of the directory information
        ii. Structure of the file identification in the directory
        iii. Structure of the location of file components (i.e., physical location of file blocks)
        iv. Method of describing the path to a file.
    h. The file system is to have a **logical directory** that describes the files and meta data as noted above, and a **physical directory** that organizes the blocks or disk units allocated to each file. The specifics of these directories and allocation methods are to be of **your design**.
        i. The physical directory will keep track of the blocks of disk space allocated to your disk itself, as well as each file in the system.

2. **You are to design the basic functions of an IOCS (Input/Output Control System)**; The I/O functionality you are developing may be considered to be the logical I/O module that deals with the device as a logical resource on behalf of users. As discussed later, you will not be responsible for writing the actual low-level device control or device scheduling. You will make use of the operating system's basic, existing

I/O functionality (e.g., the C I/O library) or using memory map files (mmap functionality). Your design will include:

   a. the parameters for each of the basic IOCS functions,
   b. the supporting data structures required in memory as part of the IOCS to implement these basic functions, and
   c. the method of synchronizing the ‘disk’ and memory representations of the directories. The basic file system functions are to include:
      i. Create file
      ii. Open file
      iii. Delete file
      iv. Close file
      v. Read file
      vi. Write file

3. **Implement your design for the file system and the IOCS functions in either the Windows environment or the Linux environment (your choice)**.
   a. You should create a file of at least 2 MB to implement your file system (you would write a program to create the file). We are posting several files (10MB, 5MB, 3MB, 2MB) that you can download and use if you choose not to create your own.
4. You are to use the operating systems file system facility for file I/O in the implementation of your file system, or use the memory mapped file system. For example, you can open the file that represents your disk and then use file read, file write and file positioning operations to work within your logical disk. You can do this using the stream I/O functions available through the C library (e.g., stream I/O, consider the block I/O), or the C low-level I/O functions. If you choose to use **memory mapped files**, then you can open the virtual disk file and map the file to memory using the mmap() system call and facility. The memory mapped file facility lets you address locations in the file and read and write the locations as if you are directly addressing memory locations. In effect, the various locations in your virtual disk are mapped to memory cells. Reading and writing to disk is managed by the OS as you read and write the memory cells. There are many references for mmap programming, including:
   Linux System Programming (Love)
   The Linux man pages for mmap

5. **Demonstrate that your file system works by writing an application** to execute in your OS and that can work with your file system: i.e., the application will at least:
   a. create a directory below the root of your file system,
   b. create a file in that directory,
   c. write a specified formatted data set into the file, and
   d. close and then reopen the file and read back and compare the data that have been written to the file.
6. **Some References for File Systems currently in Use**
   a. File System Wiki
   http://en.wikipedia.org/wiki/file_system#File_systems_under_Unix-like_operating_systems
   b. FAT File System:
      i. Design of the FAT file system
      https://en.wikipedia.org/wiki/Design_of_the_FAT_file_system
      ii. **FAT Wiki** http://en.wikipedia.org/wiki/File_Allocation_Table
      iii. http://www.ntfs.com/fat-systems.htm
      iv. http://www.pctechguide.com/31HardDisk_File_systems.htm
      v. http://msdn.microsoft.com/en-us/library/aa914353.aspx

c. NTFS File System
    i. http://www.ntfs.com/
    ii. http://en.wikipedia.org/wiki/NTFS

d. Unix File System
    i. Unix Fast File System (https:/www.cs.berkeley.edu/~brewer/cs262/FFS.pdf)
    ii. Anatomy of Linux journaling file systems (http://www.ibm.com/developerworks/library/l-journaling-filesystems/)
    iii. Next-generation Linux file systems: NiLFS(2) and exofs (http://www.ibm.com/developerworks/linux/library/l-nilfs-exofs/index.html)

e. General Information on a File System Design
    i. Practical File System Design
        1. (http://www.nobius.org/~dbg/practical-file-system-design.pdf)