CS 377
Project Report
Theme B: File Systems

Team B3

Mayank Dehgal, 110050024 Shivam H Prasad, 110050041
Aditya Raj, 110050025 Mayank Meghwanshi, 110050012
Alok Yadav, 110050043 Tanmay Randhavane, 110050010

CONTENTS

1. Overview

The aim of this project is to design and implement a File System for the Pranali OS.

We were able to implement and test the following aspects of the file system.

- Adding a library of system calls to perform
    - Creation and deletion of files and directories,
    - Open and close files,
    - Read from a file and write to a file,
    - Seek into a file and tell the position of seek pointer,
    - Change current directory.
- Store and manage directories and files.
- Allocation of disk blocks to files.
- Coarse grained protection for users' files.
- Bytestream files with indexed allocation.
- Disk cache with LRU replacement strategy.

In addition to implementing above aspects we also have added error checking in case of undesirable situations (e.g. opening a file to read which does not exist).

2. Data Structures

These are the new data structures we used in our implementation of the file system -

- **super_block** : This data structure is designed to handle the super block in the file system. It contains following variables -
    - number_of_blocks - Total number of data blocks in the disk
    - block_size - Size of a block in the disk (4096 in our case)
    - size - Maximum possible number of blocks required to store this data structure
    - FCB_root - FCB of the root directory
    - free_block_count - Blocks till this value have been assigned to the files (intermittent blocks may still be free but no block after this will be in use)
    - highest_FCB_index_used - FCB indices till this value have been assigned to the files (intermittent indices may still be free but no FCB after this will be assigned)

    - free_block_pointer - List of free blocks
    - free_FCB_list_head - List of free FCBs
- **FCB** : This data structure is designed to handle file control block of the files and directories. It contains following variables -
    - index - Unique index used to identify the FCB
    - file_size - Size of the data in this file or directory
    - name - Name of the file
    - path - Absolute path to the file
    - creation_time - Creation time of the file
    - last_modified_time
    - last_seen_time
    - uid - Author ID
    - permission - Permission id of the file to rest of the users (0 - No access, 1 - Read access, 2 - Write access)
    - block_address - File Map Table (FMT) of the file/directory
    - seek_pointers - Current seek block and an offset in that block
    - location - Location of the current file/directory in the parent directory
- **FCB_list** : This data structure is designed to implement list of FCBs used in the file system.
- **oft_entry** : This data structure is designed to model the entry of open file table. It contains following variables -
    - pid - Process id of the process opening the file
    - file_fcb - FCB of the file
    - offset - Process specific seek offset
    - mode - Read or write mode
- **cache_entry_list** : This is a list of disk cache entries. It is used to model the disk cache.
- **cache_entry** : This data structure is used to model the entry in the disk cache. It contains following variables -
    - physical_address - Address of the block in the list
    - block_data - Data in the block
    - dirty_bit - To check if the cache block is written or not
- **oft_table** : This is an array of oft_entry data structures.
- **int_list** : This is a list of int variables. It is used to store free block lists and free fcb lists in the super block.

Existing data structures in the Pranali OS that were updated -

- **ctx_t** : Added a variable path that has the current directory of the process.

3. High Level Design Overview

The file system is composed of following modules -

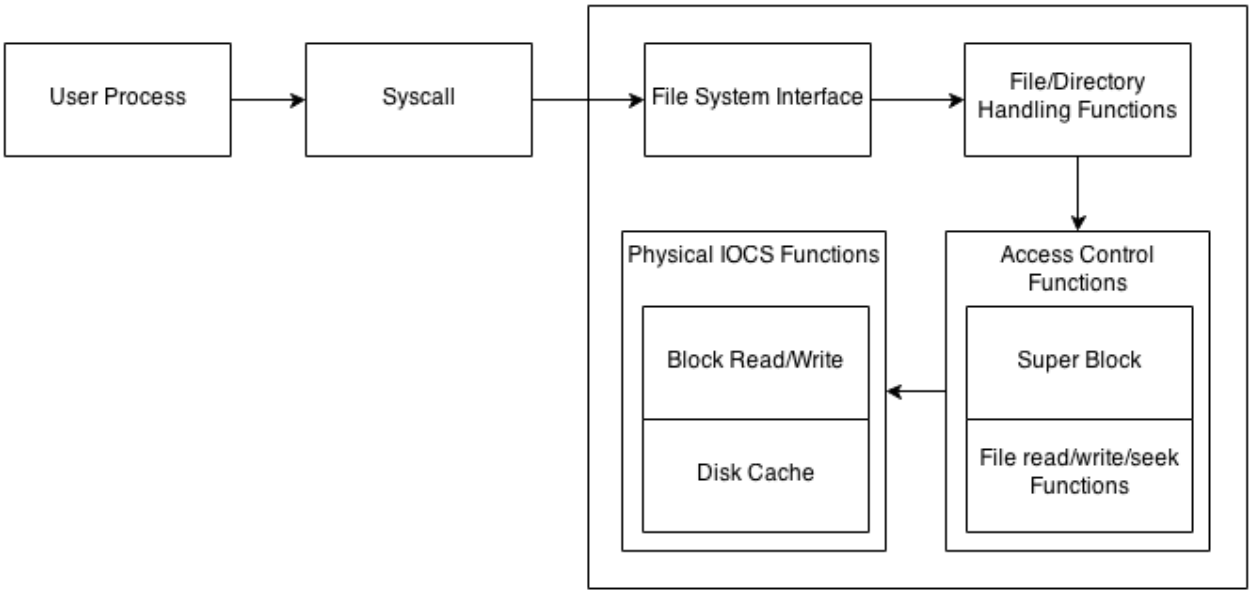

Fig. Design Overview

4. Submodule Details

4.1. System Calls

- System calls call the relevant functions from the file system interface.
- Following system calls were added -

Syscall	Input
open_file	File Path, Mode (Read/Write)
close_file	OFT index of opened file
read_file	Index, Destination Address, Size
write_file	Index, Source Address, Size

seek_file	Index, size from origin
tell_file	Index
create_directory	Path of directory
remove_file	Path of file
change_directory	Path of directory
change_permission	Path of file, Mode(No access, Allow read, Allow write)

4.2. File System Interface

- This module contains the procedures for the interface between system calls and the file system.
- Following procedures are part of file system interface -
    - open_call(path, mode, pid, uid);
        - This function deals with opening of files in different modes.
        - Read Mode: Checks if file exist, Adds it in OFT, Returns index
        - Append Mode: If file does not exists creates file, Adds it in OFT, Returns index
        - Write Mode: If file does not exists creates file, Truncates File, Adds it in OFT, Returns index
    - close_call(num, pid);
        - Updates directory entry corresponding to the file
        - Removes file from open files table
    - read_call(num, buf, size, pid);
        - Seeks file to offset corresponding to the process
        - Reads size amount of data in buf
    - write_call(num, buf, size, pid);
        - Seeks file to offset corresponding to the process
        - Writes size amount of data to buf
    - seek_call(num, size, pid);
        - Seeks file to size bytes from start of file
    - tell_call(num, pid);
        - Returns current seek offset of file

- create_directory(path, uid);
  - Check if directory exists then does not create
  - Otherwise calls file creation function for directory
- remove_call(path, uid);
  - Checks for existence of file, deletes it if exists

4.3. File/Directory Handling Functions

- This module contains functions to handle the directories and files in the file system.
- Following procedures are part of this module -
  - update_time_stamps(file, mask);
    - This function updates timestamps in FCB of file
    - Mask = 1 - Creation time, Modification time, Last seen time
    - Mask = 2 - Modification time, Last seen time
    - Mask = 3 - Last seen time
  - search_in_directory(directory, name, uid);
    - Searches for a particular entry (file or directory) in a directory and returns its FCB
  - search_file_or_directory(path, uid);
    - Returns FCB entry corresponding to the input path
  - get_parent_directory(path, uid);
    - Returns FCB of parent directory of the entry corresponding to input path
  - create_root_FCB();
    - Creates FCB for root directory
    - Permission of root is <Allow Write> i.e. anyone can modify it, but deletion of root is not allowed
    - FCB index (unique index given to a file) of root is 0
    - uid given to root is 0 but user with uid too isn't allowed to delete it
  - create_file(path, type, uid);
    - Searches for directory entry corresponding to input path
    - If found then new FCB is created for file and returned
    - type = 0 for directory, type = 1 for file
    - uid = stores owner information
    - Permission for newly created file / directory is <Allow Write> by default

- delete_file(path, uid);
  - Searches for FCB entry corresponding to input path
  - If FCB is root or permissions aren't enough delete fails
  - Once file is deleted its parent and parent of parent directories (if existing) are updated
- remove_directory(path, uid);
  - Searches for FCB entry corresponding to input path
  - If FCB is root or permissions aren't enough delete fails
  - If FCB retrieved is a file the delete_file is called
  - Recursively deletes all the contents of the directory and then calls delete_file on path only if all the contents of directory were successfully deleted (considering access permissions)
- truncate_file(file_fcb);
  - De-allocates all blocks allocated to the file and updates file size
- update_directory_trace(file);
  - Once a directory is deleted this function updates FCB entries of its parent and parent of parent (if exists)
  - Called from delete_file

4.4. Access Control Functions

- **Super Block**
  - This module implements superblock for maintenance of file system on restart of OS and contains information about disk.
  - read_super_block();
    - Reads super block struct from Sim_disk
    - Reads free block list
    - Reads free fcb list
  - init_super_block();
    - If Sim_disk does not exist then new superblock is created
    - It initialises superblock on basis of configuration file and initial conditions
  - write_super_block();
    - This function writes superblock struct in file
    - Writes updated free FCB list and free block list
  - get_free_FCB_index();

        - If free fcb list is not empty then removes one and returns
        - If list empty then adds one to highest used index and returns
    - get_free_block();
        - If free block list is not empty then removes one and returns
        - If list empty then adds one to highest used free block and returns
    - add_free_block(block_num);
        - Adds a block num to free block list
    - add_free_FCB(fcb);
        - Adds a free fcb index to free fcb list
- **File read/write/seek functions**
    - read_file(file_fcb, size)
        - Read ‘size’ number of bytes from the current seek position which is specified in the file fcb.
        - Update the seek position to current position plus size bytes in the file fcb.
    - write_file(file_fcb, size, data)
        - Write ‘size’ number of bytes from the current seek position which is specified in the file fcb.
        - This function overwrites any previously present data at that position.
        - If data was written at the end of the file, new blocks are allocated to the file as required.
        - Update the seek position to current position plus size bytes in the file fcb. Also update the file size in the fcb if changed.
    - seek_file(file_fcb, logical_address)
        - Change the current seek pointer in the file fcb to point to the new logical address in the file_fcb.
    - get_block_address(file_fcb, block_number)
        - Given a logical address of a block (block_number), translate it to physical address using the file map table in the file fcb.

4.5. Physical IOCS Functions

- **Block read/write**

    - This module implements reading and writing of disk blocks from Sim_disk file.
    - read_block(block_number);
        - Seeks the file pointer of sim disk to corresponding position.
        - Returns a buffer with size equal to block size.
    - write_block(block_number, data);
        - Seeks the file pointer of sim disk to corresponding position.
        - Writes the data in the disk block on sim_disk.
- **Disk Cache**
    - To improve the efficiency of the access of data blocks from the disk, we use disk cache.
    - LRU replacement policy is used to for cache entry replacement.
    - The disk cache is implemented as a list of cache entries.
    - Whenever a cache entry is accessed, it is removed from its original position in the cache and added at the end of the list.
    - This way, the least recently used entry is at the start of the cache list. While replacing we remove this entry and bring in the new entry at the end of the cache list.
    - remove_entry() :
        - Removes least recently accessed, i.e., the first entry of the cache list.
        - If the dirty bit is set, then the data is written back.
    - read_block_from_cache(physical_address) :
        - Given a physical address, search if the block is present in the cache. I
        - f it is present, then return the block data, otherwise, load the block from the disk, add it to cache and then return the block data.
    - init_cache() :
        - Initialize the cache entry parameters.
    - clear_cache() :
        - Remove all the cache entries from the cache list.
        - Write back if the dirty bit is set.
    - write_block_to_cache(physical_address, data) :
        - Given a physical address, search if the block is present in the cache.

- If it is present, then write the input data in the cache entry, otherwise, load the block from the disk, add it to cache and then write the block data.

5. Future Additions

5.1. Insertion into a file

- Currently, the write function in our file system overwrites any existing data at the seek position.
- We can add insert write function which will add given data to the seek position and shift the existing data ahead.
- To implement this, we can add the existing data in the block just ahead of the seek pointer to a new block. Now write the input data at the seek pointer. If this block is filled up then more blocks are allocated to this file and data is written in these blocks. We can use some delimiter to indicate the end of valid data in a block. In the end the file map table will have to be updated.
- The read and write functions will need to modified to allow for holes in the data in the blocks.

5.2. Using multiple image mutable files

- Currently, out file system implements single image mutable file sharing mechanism. We can instead use multiple image mutable files.
- In this case, when file is being written by multiple users, a copy of FCB is made for each user and for each user different copy of the file is created. When file is closed, the two copies can be merged according a specific order in the writes for the multiple users.

5.3. Reliability on crash

- Currently, no data recovery techniques are implemented for providing reliability against crashes of the Pranali OS.
- Frequent write backs of the meta data currently in memory will help to add some reliability in the file system.
- Emptying of cache happens only in case of regular shutdown of the Pranali OS. We can have frequent write backs for cache to guard against OS crashes.

5.4. Medium Granularity in protection

- Currently, we have used coarse grained protection.

- We can also implement medium granularity of protection. To do this, we can implement Access Control List (ACL) or Capability List.
- We can store this capability list or ACL as a special file in the sym_disk. At boot time, this file will be loaded into memory and will remain in the memory as long as the OS is on.
- Changing the permission of a file is allowed only through kernel system call. Permissions of only the files for whom a user has a write permission can be changed by that user.

5.5. Extensive system call library

- Currently, we have provided only the basic system calls like creation, deletion, read and write.
- We can make this library a little bit rich by adding some more syscalls which might be necessary. For example, ls and its variants, cp, mv, pwd, rename, etc.

6. Work Distribution

6.1. Mayank Meghwanshi and Aditya Raj - Super Block, System Calls

6.2. Shivam H Prasad and Mayank Dehgal - File/Directory Handling Functions, File System Interface

6.3. Tanmay Randhavane and Alok Yadav - Disk Cache, Block read/write, File read/write/seek functions

7. Comments

The project was an enjoyable task to do. We got a very good understanding into the insights of the working of the file system. We got to know some important practical things like how to make a good decision choice. Our C knowledge improved as well.