CS518 Design of Operating System
Fall 2022

Report

A2: Write-once file system

VISHWAS GOWDIHALLI MAHALINGAPPA
vg421

VIKRAM SAHAI SAXENA
vs799

Design and Implementation of writeonceFS library.

The filesystem will exist entirely inside one 4MB file in the native operating system. On mount, the library will open the file that holds the 'disk' and read all 4MB of it into memory. For the lifetime of the Process that mounted the filesystem, All the reads and writes are performed using the image in the memory. When the filesystem is unmounted, the library will write out the entire 4MB image of the modified 'disk' to the same file it came from, overwriting the old information and preserving any changes made.

Metadata Implementaion:

The file system contains a data region, an i-node region, and a superblock. However, the 'disk' supports at least 50 files and 2MB of user data. The 'disk' will not grow or shrink beyond 4MB in size. In order to emulate an actual disk, all operations are done in 1K 'disk block' quanta. We can only read or write to or from memory representing the 'disk' in 1K chunks.

super_block: This block contains directory info and data info.

i-node: This block contains file name, file size, first data block, number of blocks and a flag to check whether the file is being used or not.

The first block contains super block, second block contains i-node, and the following blocks refers to data region.

int wo_mount(char <filename>, void* <memory address>)*

filename: the 4MB file in the native operating system that holds the entire 'disk'.
memory address: an address to read entire 'disk' in to (an allocation of at least 4MB)
return int: 0 on success, any negative number on error

- *Attempt to read in an entire 'diskfile'.*
- *Return 0 on success, with any negative number representing error.*
- *On a file access error, return the error number received and set errno.*
- *If mount finds the 'disk' is blank, build initial structures for accessing the disk.*
- *If the disk format is broken, report the error and free the structures built.*

int wo_unmount(void <memory address>)*

memory address: an address where the entire 'disk' is located (an allocation of at least 4MB)
return int: 0 on success, any negative number on error

- *Attempt to write out an entire 'diskfile'.*
- *Return 0 on success, with any negative number representing error.*
- *On a file access error, return the error number received and set errno*

int wo_open(char* <filename>, <flags>)
int wo_open(char* <filename>, <flags>, <mode>)

filename: the name of a file to open the filesystem

flags: access requirements for opening one of: WO_RDONLY, WO_WRONLY, WO_RDWR

mode: switch to allow file creation. must be WO_CREAT

return int: 0 on success, any negative number on error

1. *Not in file creation mode*

- *On open, if not in file creation mode, attempt to find file.*
- *If it does not exist, return the appropriate error code and set errno.*
- *If it does exist, check permissions.*
- *If they are not compatible with the request, return the appropriate error code and set errno.*

2. *In file creation mode*

- *If in file create mode, check to make sure file does not exist*
- *if it does, return the appropriate error code and set errno.*
- *If it does not exist, create an entry for it and create and return a file descriptor for it.*

int wo_read(int <fd>, void <buffer>, int <bytes>)*

fd: a valid filedscriptor for the filesystem

buffer: a memory location to either read bytes in to or frombytes

bytes: the number of bytes to read or write.

return int: 0 on success, any negative number on error

- *Check to see if the given filedscriptor is valid or has an entry in the current table of open file descriptors.*
- *If not, return with error and set errno.*
- *If valid, perform the read operation, read bytes from the file indicated to the location given.*

int wo_write(int <fd>, void <buffer>, int <bytes>)*

fd: a valid filedscriptor for the filesystem

buffer: a memory location to either read bytes in to or from bytes

bytes: the number of bytes to read or write

return int: 0 on success, any negative number on error

- *Check to see if the given filedscriptor is valid or has an entry in the current table of open file descriptors.*
- *If not, return with error and set errno.*
- *If valid, perform the write operation, write bytes from the location given to the file indicated.*

int wo_close(int <fd>)

fd: a valid filedscriptor for the filesystem

return int: 0 on success, any negative number on error

- *Check if the given filedescriptor is valid or has an entry in the current table of open file descriptors.*
- *If not, return with error and set errno.*
- *If valid, mark it as closed (remove from the current table of open file descriptors).*

*int ready_disk(char *file_name)*

- *Ready a disk for open/create from File System.*
- *@param file_name: File System file name*
- *@return int: 0 on success, any negative number on error*

*int open_disk(char *file_name)*

- *Open a disk from File System.*
- *@param file_name: File System file name*
- *@return int : 0 on success, any negative number on error*

int close_disk()

- *Close the disk representing the File System*
- *@return int: 0 on success, any negative number on error*

*int read_block(int block_index, char *buffer)*

- *Read block contents to buffer*
- *@param block_index: block index*
- *@param buffer: buffer*
- *@return int: 0 on success, any negative number on error*

*int write_block(int block_index, char *buffer)*

- *Write contents from buffer to block*
- *@param block_index: block index*
- *@param buffer: buffer*
- *@return int: 0 on success, any negative number on error*

*int disk_init(char *file_name, void *mem_address)*

- *Initialize structures for accessing disk.*
- *@param file_name: File System file name*
- *@param mem_address: address to read the entire 'disk' in to*
- *@return int: 0 on success, any negative number on error*

char search_file(char file_name)*

- *Search for a file in the File System disk*
- *@param file_name: file name to search*
- *@return char: file index on success, any negative number on error*

int available_file_des(char file_index)

- *Fund available file descriptor from file descriptor table*
- *@param file_index: file index*
- *@return int: file descriptor on success, any negative number on error*

int search_next_block(int current_block_index, char file_index)

- *Search next block of a file*
- *@param current_block_index: current block index*
- *@param file_index: file undex*
- *@return int: next block index on success, any negative number on error*

int search_next_block(int current_block_index, char file_index)

- *Search next block of a file*
- *@param current_block_index: current block index*
- *@param file_index: file undex*
- *@return int: next block index on success, any negative number on error*

int search_available_block(char file_index)

- *Search available data block for writing file*
- *@param file_index: file index*
- *@return int: available data block number on success, any negative number on error*

*int wo_create(char *file_name)*

- *Create a file in the File System disk if mode is WO_CREAT*
- *@param file_name: file name to create in the File System*
- *@return int: 0 on success, any negative number on error*

Test Results

```
disk_init()       called successfully.
wo_create()       called successfully: file [test.txt] created.
wo_open()         called successfully: file [test.txt] opened.
wo_open()         called successfully: file [test.txt] opened.
wo_close()        called successfully: file [test.txt] closed
wo_close()        called successfully: file [test.txt] closed
wo_open()         called successfully: file [test.txt] opened.
wo_read()         called successfully.
```

Make file run:

```
make clean
make all
./test
```