

# Project Report

## CSE 536

Name: Vishal Srivastava

ID No.:1209824652

### Assumptions

Under the following assumptions the program works fine:

1. All access rights are available with the required directory structure.
2. Each cell holds no more than one data from a particular file.
3. Each cell block has a size of 1024 bytes and can store value as strings.
4. Each cell is owned by a particular file and thus cannot be accessed or written by another file.
5. The program writes on one cell at a time with its associated file id.
6. The program can only store 100 cells in its temporary storage (state before commit).
7. All blocks/cells belonging to a certain file are committed simultaneously
8. Several cells/block of files need not be concurrent to each other.
9. Currently the system is neither fault tolerant nor is multithreaded

### Implementation Procedure

Currently for Phase I, the whole program is divided into 3 main header files namely cellstorage.h, journal\_file\_manager.h and storageunits.h. A structure defined for storage and is written and read from files. Each text file under the cells folder is a cell block where structure block is written and read from.

### Cell Storage

The cell storage contains all the associated functions like IS\_OWNER, CHECK\_FOR\_BLOCK, READ, WRITE, ALLOCATE and DEALLOCATE.

1. IS\_OWNER  
Use to check for the ownership for that particular block with the supplied file\_id.  
INPUT: int file\_id and int cells  
OUTPUT: Returns 1,2,3 for OK, Failure to read and Access violation
2. CHECK\_FOR\_BLOCK  
Use to check if a certain block is available in the cells directory or not.

INPUT: String filename (block no)

OUTPUT: Returns 0,1 for True and False in terms of availability

3. READ

Use to read data from the block/cell associated with file id.

INPUT: int file id, int process id (p\_id) and int cell no

OUTPUT: Returns string containing the data stored in the cell

4. WRITE

Use to write data to the cell block with an associated file id and stores data in the cell.

INPUT: int file id, int process id (p\_id), int cell\_no and string value\_data

OUTPUTS: Returns 0,1 for OK, and Error

## Journaling File Manager

Uses the implemented storage block for the file system for temporary storage of data. It keeps a state for temporary storage of data before commit and keeps the data on a non-volatile storage. Acts as an abstraction layer over the Cell storage system. It uses the following programs:

1. NEW\_ACTION

Allocates a cell block to a particular file id and creates a process\_id

INPUT: int file\_id

OUTPUT: Returns a integer type generated process\_id

2. READ\_CURRENT\_VALUE

Reads a value from the cell storage with the associated cell and file id

INPUT: int file id, int process id, int cell no

OUTPUT: Returns string of data from that particular cell/block.

3. COMMIT

Use to commit data associated with a particular file id to the cell storage system. It retrieves the data from the temp file and retrieves the temporary storage queue. Scans and Searches for data for a particular file\_id and process id and valid field. If valid is 1 then the data is committed for the associated file id and process id. After committing make the valid entry to 0.

INPUT: int file\_id, process\_id

OUTPUT: returns 0,1 for OK and failure

4. WRITE\_NEW\_VALUE

Used to store the data in the temporary storage before committing. It retrieves the data from the temp file and retrieves the temporary storage queue. Scans and Searches for new or invalid entry from start of the queue. Enters the data and set the valid field to 1, in the invalid or new storage block and saves it back to temp storage file.

INPUT: int file id, int process id, int cell no, string data

OUTPUT: Returns 0,1 for OK and failure

5. ABORT

Use to abort all the entry in the temporary storage associated with a particular process id and file id. It retrieves the data from the temp file and retrieves the temporary storage queue. Scans and Searches for all the possible entries associated with that process id and file id and set their valid field to 0. It again writes back to file with the changes.

INPUT: int file id and int process id

6. Cleanup

Used to reset the state of the system by deleting all the state files and temporary storage file.

INPUT: Nothing

OUTPUT: Nothing

## Test cases and reason for the choice

We have chosen the following test cases:

1. Normal read write operation

In this test case we check for normal read and write functionality of the program with the commit. Here we first allocate space using NEW\_ACTION and retrieves the generated process id. Then using process id and field id we write the data to the first block of the temp storage file. Then we store commit the values for the block to cell storage system. After committing we try to read the value using the READ\_CURRENT\_VALUE and we retrieve back the value from cell storage file and display it.

This test case checks the normal operation of the file system which is supposed to be the maximum case that can be observed.

2. Read write without commit

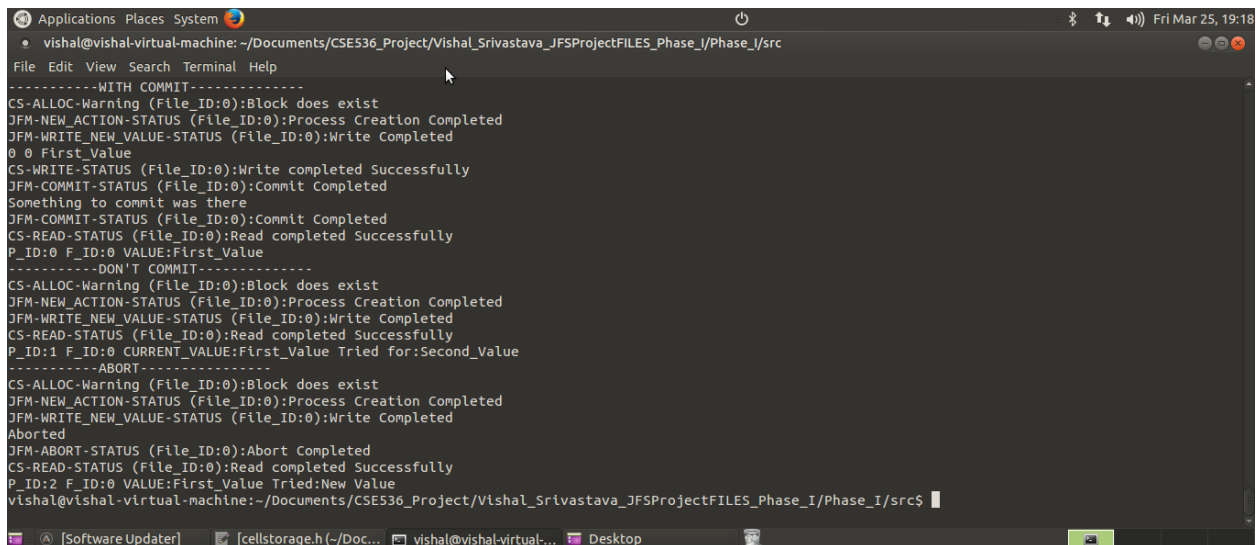
In this test case we did the same as above but just that we didn't commit after writing to the temporary storage block. After writing we again tried reading the values from the cell storage system. As we have not committed the write so the value didn't get stored in the cell storage system. Thus when we tried reading from it it gave the old value that was stored in cell storage system.

This was done to check the functionality of the commit operation without which the mutation of data is not carried forward to our permanent storage that is cell storage.

3. Read write with abort

We did the same read and write just like then earlier case but after write we put the abort to stop the mutation to be carry forwarded to the cell storage system. Abort flushes down all the entries in temporary storage associated with a valid entry to a file id and process id. Then even after the commit none of the cell blocks will get updated for that file id and process id as none of them are valid now. Then if we tried to read we will get the old data in cell for that particular file id.

## Test run screenshot and explanation



```
vishal@vishal-virtual-machine: ~/Documents/CSE536_Project/Vishal_Srivastava_JFSProjectFILES_Phase_I/Phase_I/src
File Edit View Search Terminal Help
-----WITH COMMIT-----
CS-ALLOC-Warning (File_ID:0):Block does exist
JFM-NEW_ACTION-STATUS (File_ID:0):Process Creation Completed
JFM-WRITE_NEW_VALUE-STATUS (File_ID:0):Write Completed
0 0 First_Value
CS-WRITE-STATUS (File_ID:0):Write completed Successfully
JFM-COMMIT-STATUS (File_ID:0):Commit Completed
Something to commit was there
JFM-COMMIT-STATUS (File_ID:0):Commit Completed
CS-READ-STATUS (File_ID:0):Read completed Successfully
P_ID:0 F_ID:0 VALUE:First_Value
-----DON'T COMMIT-----
CS-ALLOC-Warning (File_ID:0):Block does exist
JFM-NEW_ACTION-STATUS (File_ID:0):Process Creation Completed
JFM-WRITE_NEW_VALUE-STATUS (File_ID:0):Write Completed
CS-READ-STATUS (File_ID:0):Read completed Successfully
P_ID:1 F_ID:0 CURRENT_VALUE:First_Value Tried For:Second_Value
-----ABORT-----
CS-ALLOC-Warning (File_ID:0):Block does exist
JFM-NEW_ACTION-STATUS (File_ID:0):Process Creation Completed
JFM-WRITE_NEW_VALUE-STATUS (File_ID:0):Write Completed
Aborted
JFM-ABORT-STATUS (File_ID:0):Abort Completed
CS-READ-STATUS (File_ID:0):Read completed Successfully
P_ID:2 F_ID:0 VALUE:First_Value Tried:New Value
vishal@vishal-virtual-machine:~/Documents/CSE536_Project/Vishal_Srivastava_JFSProjectFILES_Phase_I/Phase_I/src$
```

The above test cases are tested in the same file. We can see as the blocks/cell were already there from previous run Cell Storage (CS) gave a warning for it. Then New action returned from Journal File Manager (JFM) gave an all clear status for creating the process. Then WRITE\_NEW\_VALUE successfully wrote the value. As there were some entries left to be committed the JFM searched for those entries and committed those entries. Then the CS returned back the successful commit message and the same message is replicated back to the JFM where it also returns committed successfully. Then a read is done which returned back the value and the value is printed with process Id and file id.

For the second test, is also same but we haven't committed here. So the last line given us the value (First Value) that it retrieved from the cell storage and the value ('Second Value') it tried to write to the storage system.

For the third test, which is also same as second. Here we wrote the values to first temporary storage but then we aborted and thus we removed all the values associated with file id and process id. Then we tried to read value from cell storage and found that the value (New Value) we tried and value (First Value) we got from cell storage were different.

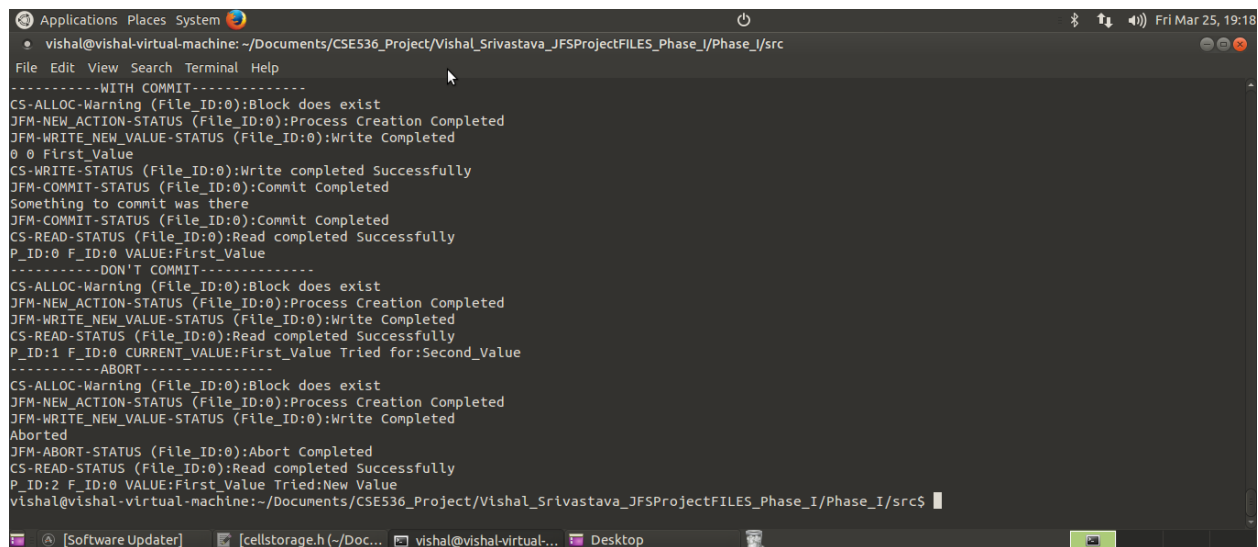
## Results

We were able to successfully implement the cell storage and Journaling File system with minor problems. The results we obtained tally with the design. As time was a factor for completing this assignment we had to be fast about it. There could be some problems with the implementation under certain test case as it's still not well tested but works well with the foreseen test cases.

## Discussion

As there wasn't enough time so couldn't divide the program but each function is independent to the rest and thus can work effectively in later phases. Also as the whole file system both temporary and cell storage is implemented in the non-volatile storage using files as blocks so in case of failure only a certain segment will be lost but the downside to this implementation is that this system will be slow for read and writes. This kind of system is generally observed in the database management system where failure will be catastrophic. So we went with this type of implementations. Later on maybe we can have cache system to it but for now it works perfectly fine.

## Output Screenshots of program running



```
vishal@vishal-virtual-machine: ~/Documents/CSE536_Project/Vishal_Srivastava_JFSProjectFILES_Phase_I/Phase_I/src
File Edit View Search Terminal Help
-----WITH COMMIT-----
CS-ALLOC-Warning (File_ID:0):Block does exist
JFM-NEW_ACTION-STATUS (File_ID:0):Process Creation Completed
JFM-WRITE_NEW_VALUE-STATUS (File_ID:0):Write Completed
0 0 First_Value
CS-WRITE-STATUS (File_ID:0):Write completed Successfully
JFM-COMMIT-STATUS (File_ID:0):Commit Completed
Something to commit was there
JFM-COMMIT-STATUS (File_ID:0):Commit Completed
CS-READ-STATUS (File_ID:0):Read completed Successfully
P_ID:0 F_ID:0 VALUE:First_Value
-----DON'T COMMIT-----
CS-ALLOC-Warning (File_ID:0):Block does exist
JFM-NEW_ACTION-STATUS (File_ID:0):Process Creation Completed
JFM-WRITE_NEW_VALUE-STATUS (File_ID:0):Write Completed
CS-READ-STATUS (File_ID:0):Read completed Successfully
P_ID:1 F_ID:0 CURRENT_VALUE:First_Value Tried for:Second_Value
-----ABORT-----
CS-ALLOC-Warning (File_ID:0):Block does exist
JFM-NEW_ACTION-STATUS (File_ID:0):Process Creation Completed
JFM-WRITE_NEW_VALUE-STATUS (File_ID:0):Write Completed
Aborted
JFM-ABORT-STATUS (File_ID:0):Abort Completed
CS-READ-STATUS (File_ID:0):Read completed Successfully
P_ID:2 F_ID:0 VALUE:First_Value Tried:New Value
vishal@vishal-virtual-machine:~/Documents/CSE536_Project/Vishal_Srivastava_JFSProjectFILES_Phase_I/Phase_I/src$
```