

**CSE 536: Advance Operating System**  
**Project: Journal Storage System**

**Assumption:**

All operations will be conducted on single block. No large file operations are required. As C programming language does not support object oriented technology and pointers concept, I developed the code using C++.

**Implementation Procedures:**

To develop journal file system, I have used concept of dynamic memory allocation which includes pointers and referencing of data object. Dynamic memory allocation means the memory is allocated to the object at runtime and because of this it precluded constrain of stating number of objects that be created and number of transactions that can be performed before compiling. Hence, a user can create any no. of data object and can perform any number of transactions with-out any limit.

For concurrent access to a file, I have used p-thread concept in C++. Here, when a write or read operation is performed by the user, a new thread is created and executed. Thread consists the data block id and action id along with value that has to been written.

For concurrency, I have develop lock mechanism. In this mechanism, I have created a data structure called lock which maintain the status of each data id. If the status of a data id is 1, that means this particular is available for the operation. If the data id is 2, that means a thread is performing a writing operation that particular thread. Each thread before writing to a particular data id changes its status in lock data structure. After performing the write operation it again changes the status of the data id to 1.

For recovery, I have implemented logs. In log mechanism, data is written to a file when a committed action takes place. In log file we have data id, action id and value. Now, during recovery, recover procedure fetches one row at a time from the starting of the file and write the value again to the system. In this way, it handle the crashes.

**Environment Specification**

- 1) System Configurations: Ubuntu Distribution 14.04.2 LTS
- 2) Programming Language: C\C++
- 3) Compiler on which application is executed: GCC

Code has six procedure to perform different operation. Following are the procedures:

- 1) New\_action() : It auto generates action id and initialize state with 1(Pending). It return action id which is an integer.
- 2) New\_value(): This procedure is responsible for writing a new value to a particular location.
- 3) Commit(): This procedure changes the state of a particular action id to 2(Committed). This means changes made by New\_value procedure is visible to all other users.
- 4) Abort(); This procedure changes state of a particular action id to 3(Aborted). This means changes made by new\_value procedure has been reverted back.
- 5) Read\_current\_value : This procedure reads the last committed value for a particular data object.
- 6) Exit(): This procedure is called to shut down the system.

## Test cases

### ID : Test Case 1

Purpose: Test all-or-nothing atomicity

Prerequisite:

1. Read, Write, Commit Procedures have been implemented.

Steps:

0. Read information at a certain location from disc.

Reading a value from data object "1". As you can see in the below screen shot. Last Committed value is **23**

```
3. commit
4. Abort
5. Read a value
6. Exit
Please Enter your choice(reversed keywords 0,1,2,3) :3

1. New Action
2. Write a value
3. commit
4. Abort
5. Read a value
6. Exit
Please Enter your choice(reversed keywords 0,1,2,3) :5

reading current value
please enter data :1
```

1. Write some information to that position in disc.

Now overwrite the same data object with another value. As you can see in the screenshot, new value to data object "1" is **"45"**

```

3. commit
4. Abort
5. Read a value
6. Exit
Please Enter your choice(reversed keywords 0,1,2,3) :1

1. New Action
2. Write a value
3. commit
4. Abort
5. Read a value
6. Exit
Please Enter your choice(reversed keywords 0,1,2,3) :2

Please enter data id :1

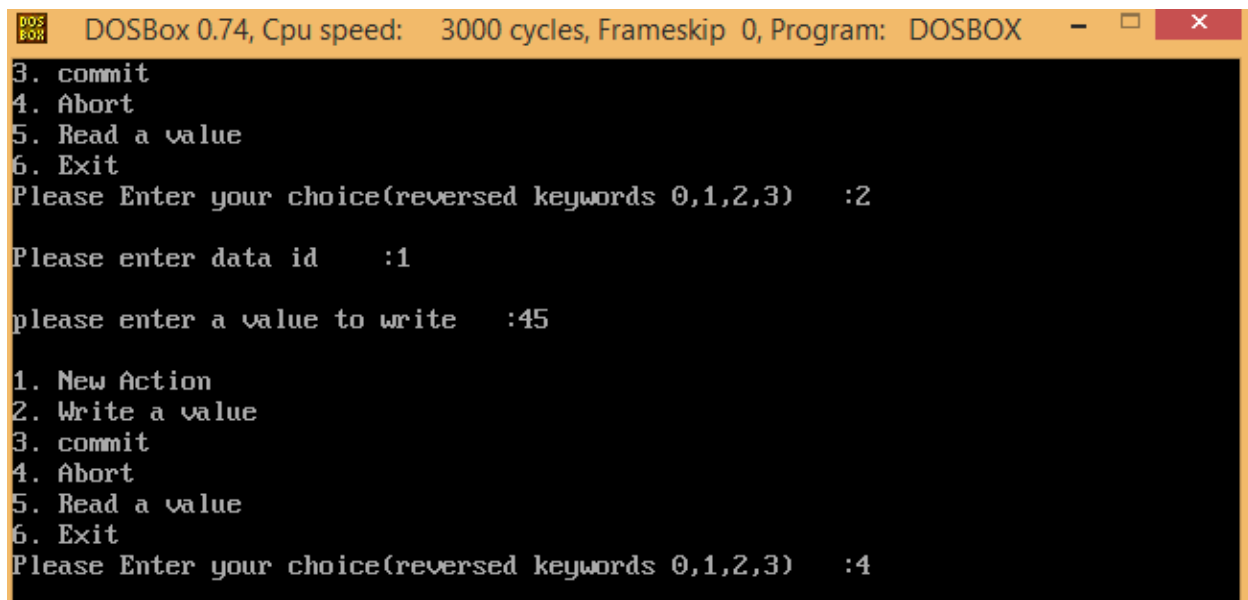
please enter a value to write :45

1. New Action

```

2. Abort the write request (i.e. do not commit)

After performing the read operation, we have to abort the last value written to data object 1. As in you can see in the diagram, Option choose was '4'. It means that the last write operation is aborted.



```

DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX
3. commit
4. Abort
5. Read a value
6. Exit
Please Enter your choice(reversed keywords 0,1,2,3) :2

Please enter data id :1

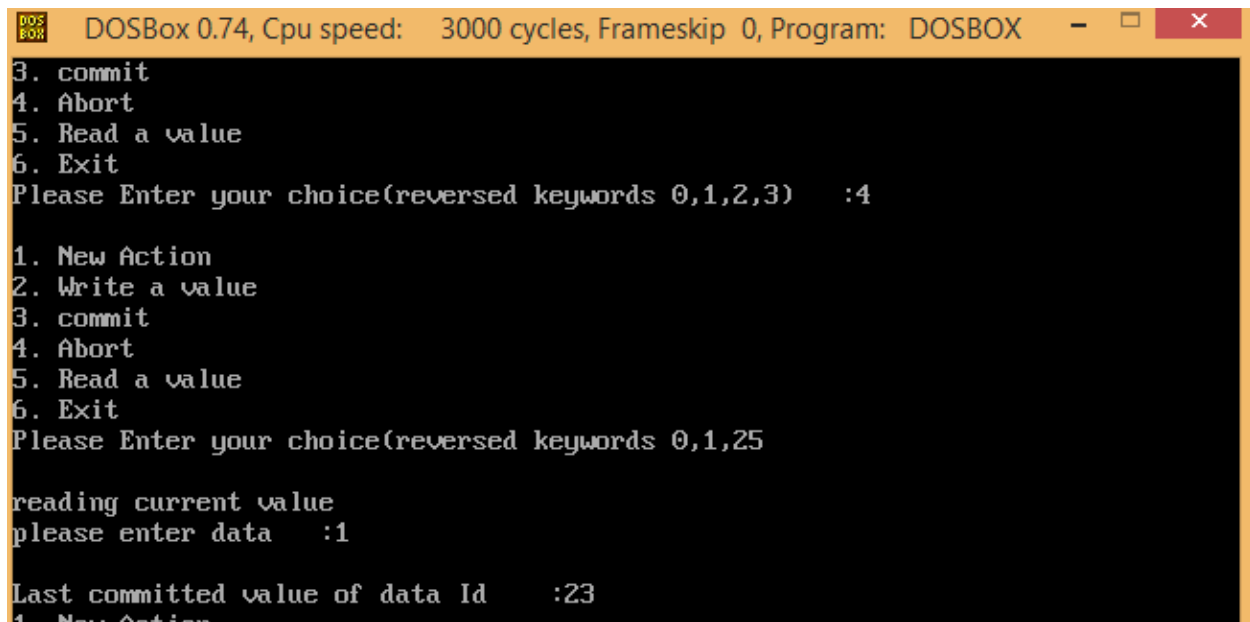
please enter a value to write :45

1. New Action
2. Write a value
3. commit
4. Abort
5. Read a value
6. Exit
Please Enter your choice(reversed keywords 0,1,2,3) :4

```

3. Read from the write location.

After all the above steps, now read the value of data object "1". As we can see in the screenshot, last committed value read is "23"



```
DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX
3. commit
4. Abort
5. Read a value
6. Exit
Please Enter your choice(reversed keywords 0,1,2,3) :4

1. New Action
2. Write a value
3. commit
4. Abort
5. Read a value
6. Exit
Please Enter your choice(reversed keywords 0,1,2,5

reading current value
please enter data :1

Last committed value of data Id :23
1. New Action
```

Expected:

1. The information that is read at Step 3. Should be the same as information in step 0.

Outcome:

PASS. As we can compare the output of step 0 and step 3 is same that is 23.

## ID: Test Case 2

**Purpose: General Read-Write**

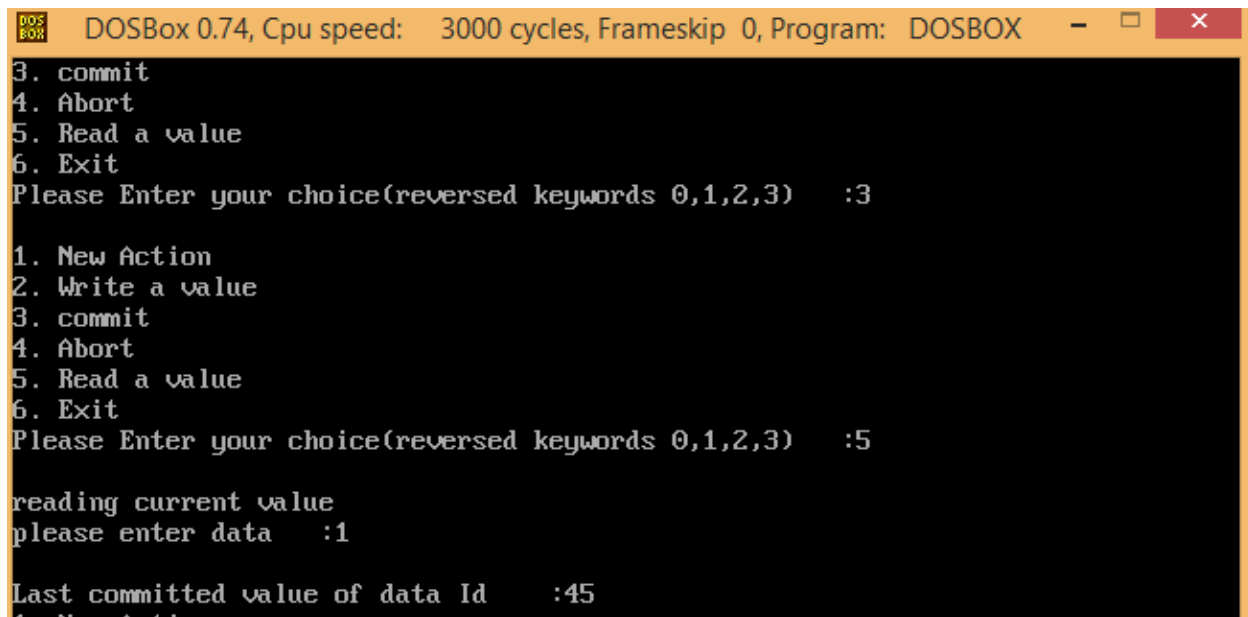
**Prerequisite:**

1. Read, Write, Commit Procedures have been implemented.

Steps:

0. Read information at a certain location from disc.

In this step, reading the last committed value for data object 1. As depicted in screenshot, the last committed value is



```
DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX
3. commit
4. Abort
5. Read a value
6. Exit
Please Enter your choice(reversed keywords 0,1,2,3) :3

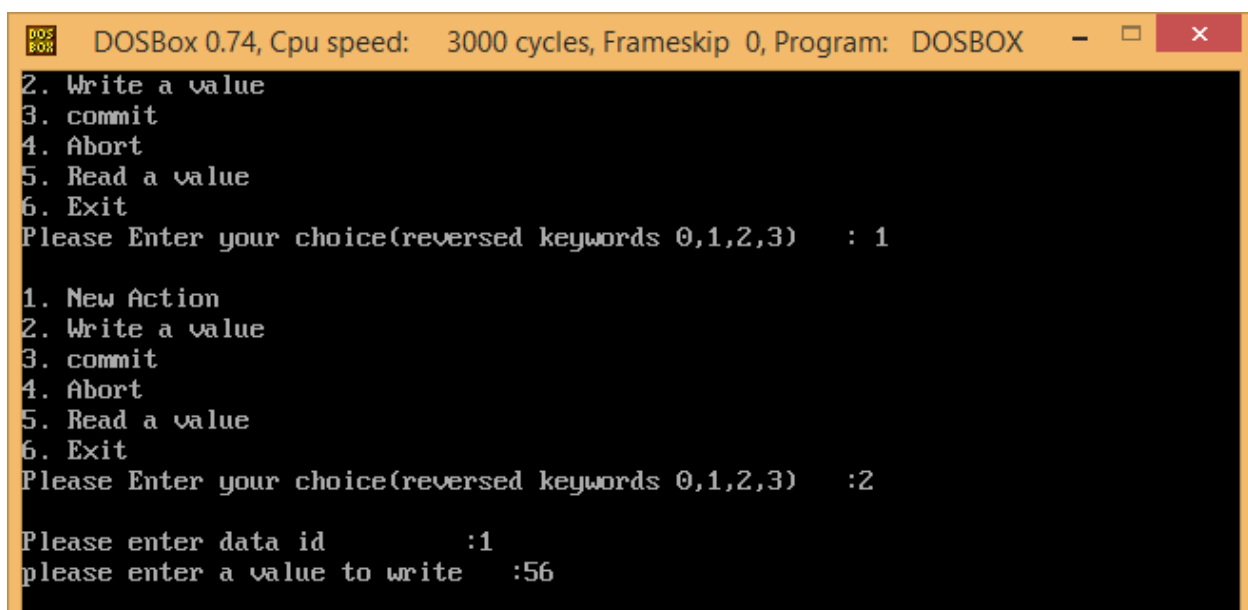
1. New Action
2. Write a value
3. commit
4. Abort
5. Read a value
6. Exit
Please Enter your choice(reversed keywords 0,1,2,3) :5

reading current value
please enter data :1

Last committed value of data Id :45
1. New Action
```

1. Write some information to that position in disc.

In this step, new value that is 56 is written to data object 1.



```
DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX
2. Write a value
3. commit
4. Abort
5. Read a value
6. Exit
Please Enter your choice(reversed keywords 0,1,2,3) : 2

1. New Action
2. Write a value
3. commit
4. Abort
5. Read a value
6. Exit
Please Enter your choice(reversed keywords 0,1,2,3) :3

Please enter data id :1
please enter a value to write :56
```

2. Commit the write request.

In this step, the action performed in the last step is committed and new value i.e. 56 is stored permanently.

```
DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX
2. Write a value
3. commit
4. Abort
5. Read a value
6. Exit
Please Enter your choice(reversed keywords 0,1,2,3) :2

Please enter data id :1
please enter a value to write :56

1. New Action
2. Write a value
3. commit
4. Abort
5. Read a value
6. Exit
Please Enter your choice(reversed keywords 0,1,2,3) :3
```

3. Read from the write location.

After commit operation, in this step, we cross verify whether new value is visible or not.

```
DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX
3. commit
4. Abort
5. Read a value
6. Exit
Please Enter your choice(reversed keywords 0,1,2,3) :3

1. New Action
2. Write a value
3. commit
4. Abort
5. Read a value
6. Exit
Please Enter your choice(reversed keywords 0,1,2,3) :5

reading current value
please enter data :1

Last committed value of data Id :56
```

Expected:

1. The information that is read at Step 3. Should be the same as information that was written.

Outcome:

PASS. As we can analyze that the value written in step 2 is same as step 3.

**ID : Test Case 3**

### Purpose: General Read Write

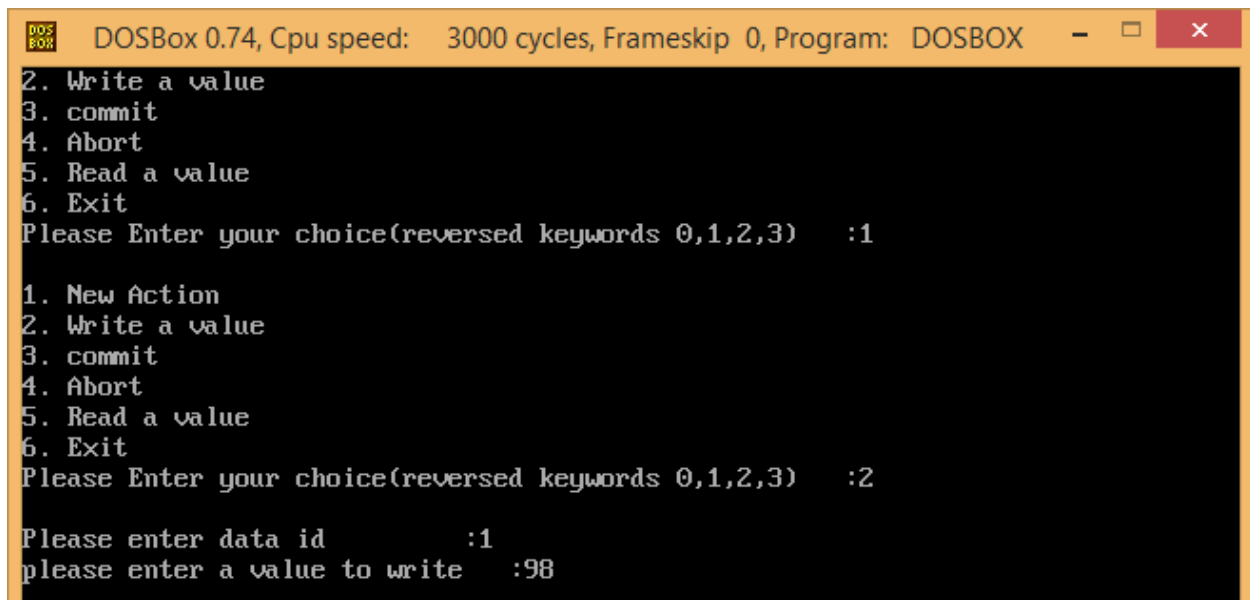
Prerequisite:

1. Read, Write, Commit Procedures have been implemented.

Steps:

1. Write some information to a position in disc.

In this operation, a new value i.e. **98** written to the data object 1.



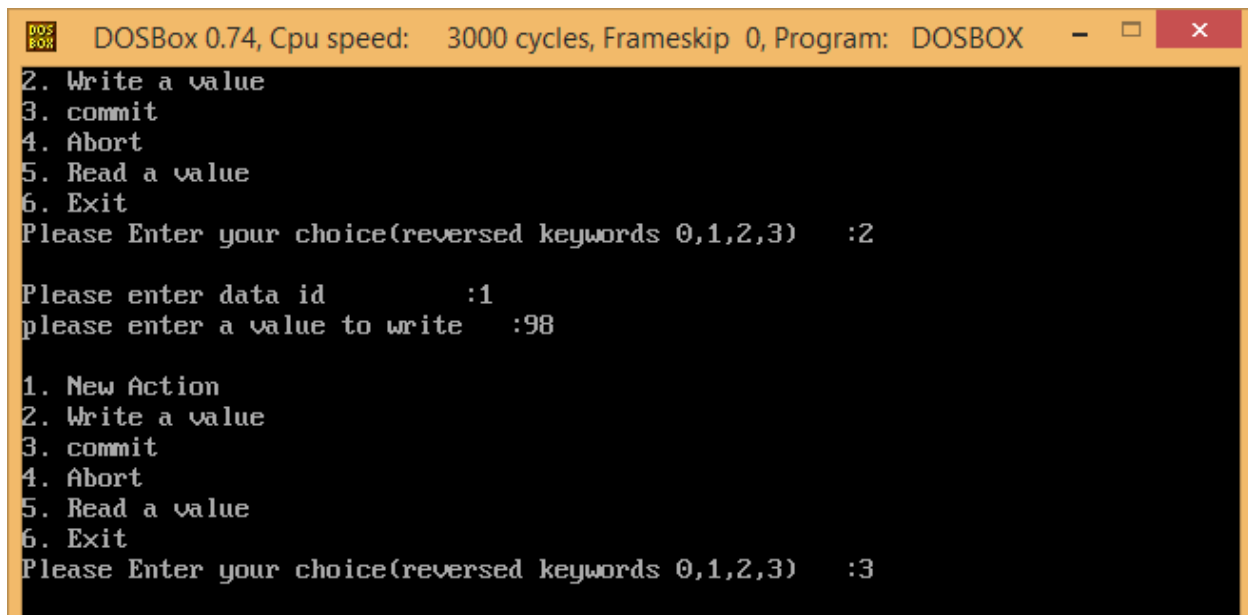
```
DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX
2. Write a value
3. commit
4. Abort
5. Read a value
6. Exit
Please Enter your choice(reversed keywords 0,1,2,3) :1

1. New Action
2. Write a value
3. commit
4. Abort
5. Read a value
6. Exit
Please Enter your choice(reversed keywords 0,1,2,3) :2

Please enter data id :1
please enter a value to write :98
```

2. Commit the write

In this step, action performed in the above step is committed.

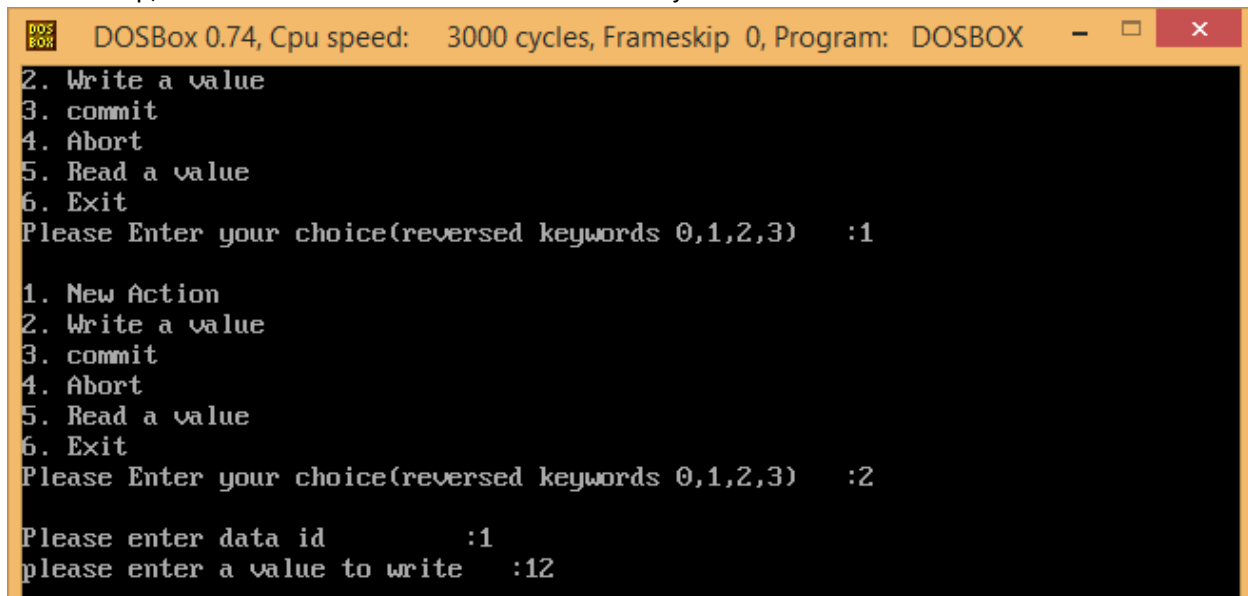


```
DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX
2. Write a value
3. commit
4. Abort
5. Read a value
6. Exit
Please Enter your choice(reversed keywords 0,1,2,3) :2

Please enter data id :1
please enter a value to write :98

1. New Action
2. Write a value
3. commit
4. Abort
5. Read a value
6. Exit
Please Enter your choice(reversed keywords 0,1,2,3) :3
```

3. Write another information in the same position (same sector and address if implemented as such)  
In this step, a new value i.e 12 is written to the data object 1.



```
DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX
2. Write a value
3. commit
4. Abort
5. Read a value
6. Exit
Please Enter your choice(reversed keywords 0,1,2,3) :1

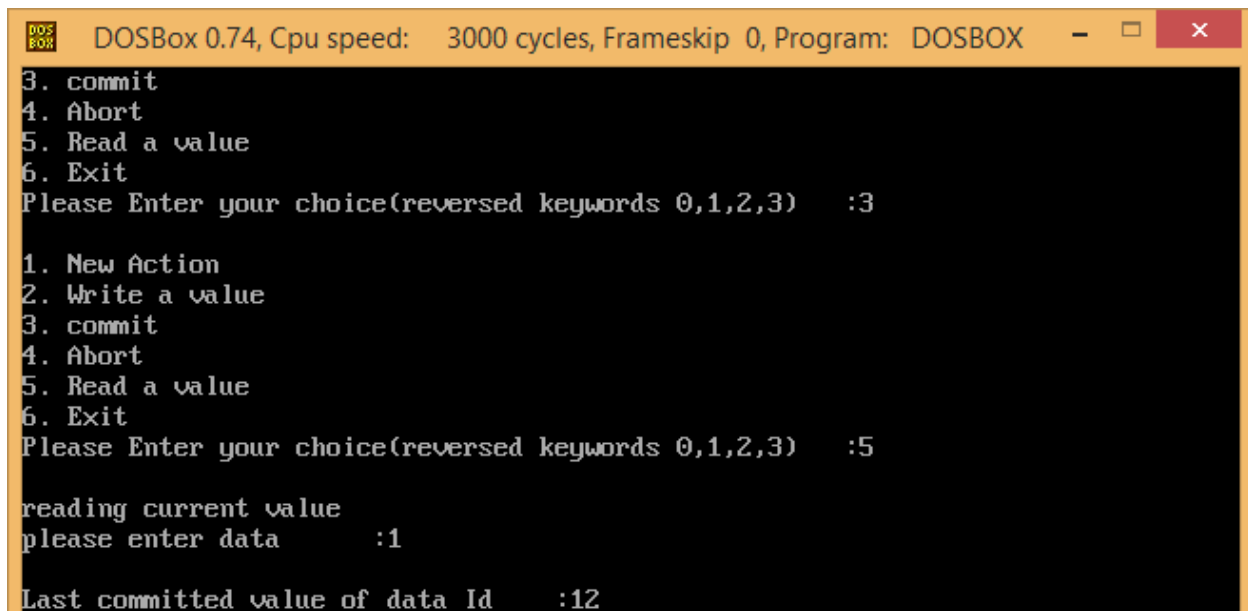
1. New Action
2. Write a value
3. commit
4. Abort
5. Read a value
6. Exit
Please Enter your choice(reversed keywords 0,1,2,3) :2

Please enter data id :1
please enter a value to write :12
```

4. Read from the write location.

In this step, last value inserted by the write operation performed in above step is read from the disc.





```
DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX
3. commit
4. Abort
5. Read a value
6. Exit
Please Enter your choice(reversed keywords 0,1,2,3) :3

1. New Action
2. Write a value
3. commit
4. Abort
5. Read a value
6. Exit
Please Enter your choice(reversed keywords 0,1,2,3) :5

reading current value
please enter data :1

Last committed value of data Id :12
```

Expected:

1. The information that is read at Step 4. Should be the result of the most recent write.

Outcome:

PASS. Value read in step 4 and value wrote in the step 3 are same.

#### ID: Test Case 4

**Purpose: Before or after atomicity**

Prerequisite:

1. Locks: Lock mechanism is implemented using Lock data structure.
2. Read, Write, Commit Procedures have been implemented.

**Command to execute this particular test case:**

**a) Write twice to a single data id that is 1.**

```
did = 1;
value = 50;
threadid++;
pthread_create(&t[threadid], NULL,&writefunction,NULL);
did = 1;
value = 60;
threadid++;
pthread_create(&t[threadid], NULL,&writefunction,NULL);
```

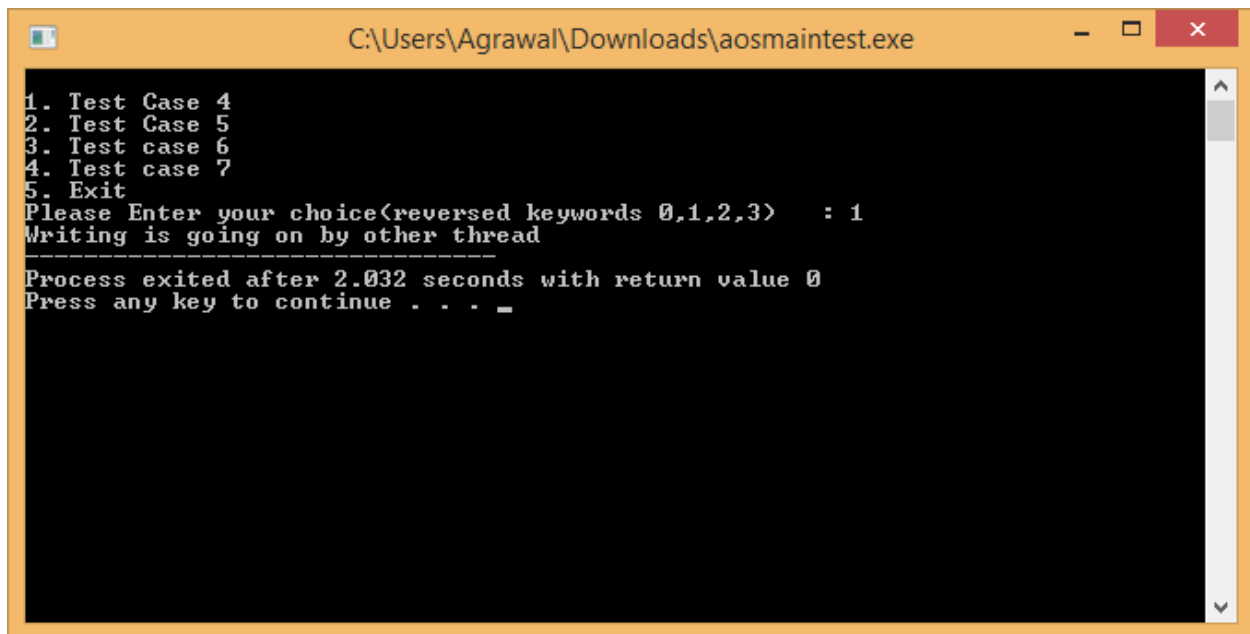
**Steps:**

0. Read the information at a certain location in a file.
1. Write information to the disk (using one thread)
2. Optional: Sleep in that thread for a time being.
3. Try to write to that location in the FS, using another thread(s)

Expected:

1. The second and subsequent threads should return unsuccessful and call your programs appropriate handler (which might decide to wait for a certain time and try to reacquire the lock)

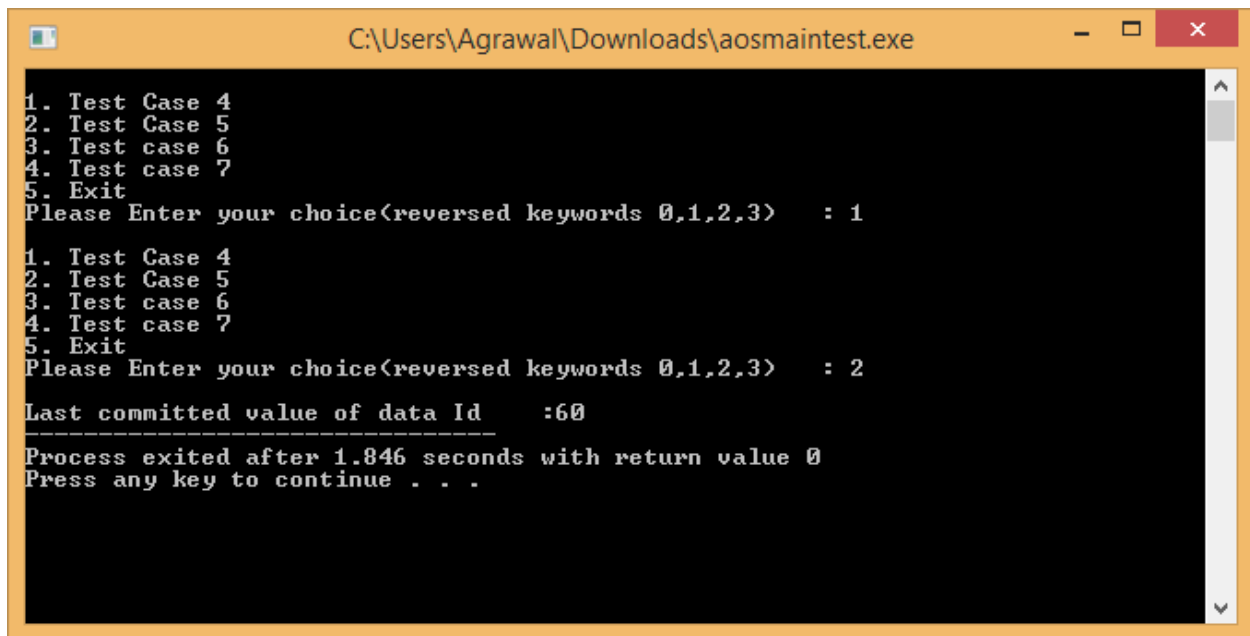
Output Snapshots.



```
C:\Users\Agrawal\Downloads\aosmaintest.exe

1. Test Case 4
2. Test Case 5
3. Test case 6
4. Test case 7
5. Exit
Please Enter your choice(reversed keywords 0,1,2,3) : 1
Writing is going on by other thread
-----
Process exited after 2.032 seconds with return value 0
Press any key to continue . . . _
```

As we can see in the output screen that once system tried to write to a particular data block concurrently, it gave an output saying that writing is going on by other thread.



```
C:\Users\Agrawal\Downloads\aosmaintest.exe

1. Test Case 4
2. Test Case 5
3. Test case 6
4. Test case 7
5. Exit
Please Enter your choice(reversed keywords 0,1,2,3) : 1

1. Test Case 4
2. Test Case 5
3. Test case 6
4. Test case 7
5. Exit
Please Enter your choice(reversed keywords 0,1,2,3) : 2

Last committed value of data Id :60
-----
Process exited after 1.846 seconds with return value 0
Press any key to continue . . .
```

Here, the value of data-id 1 is 60. That means it update data id 1.;

Outcome:

PASS

#### ID: Test Case 5

##### Purpose: Test Error-Prone Environment

Prerequisite:

1. Read, Write, Commit Procedures have been implemented.
2. Error prone read/write procedures

##### Command to execute this particular test case:

- 1) Write a value to particular dataid. Here, we are writing to dataid = 3.

```
did = 3;
value = 150;
threadid++;
pthread_create(&t[threadid], NULL,&writefunction,NULL);
```

- 2) Then close whole program. It will save the value to a file called **LogEntry**.  
Manually close the window.
- 3) Re-execute the program with recovery method.  
recovery(ha,hl,h);
- 4) Now read the value for the data id = 3

```
didread = 2;  
read_current_value(ha,h,didread);
```

Steps:

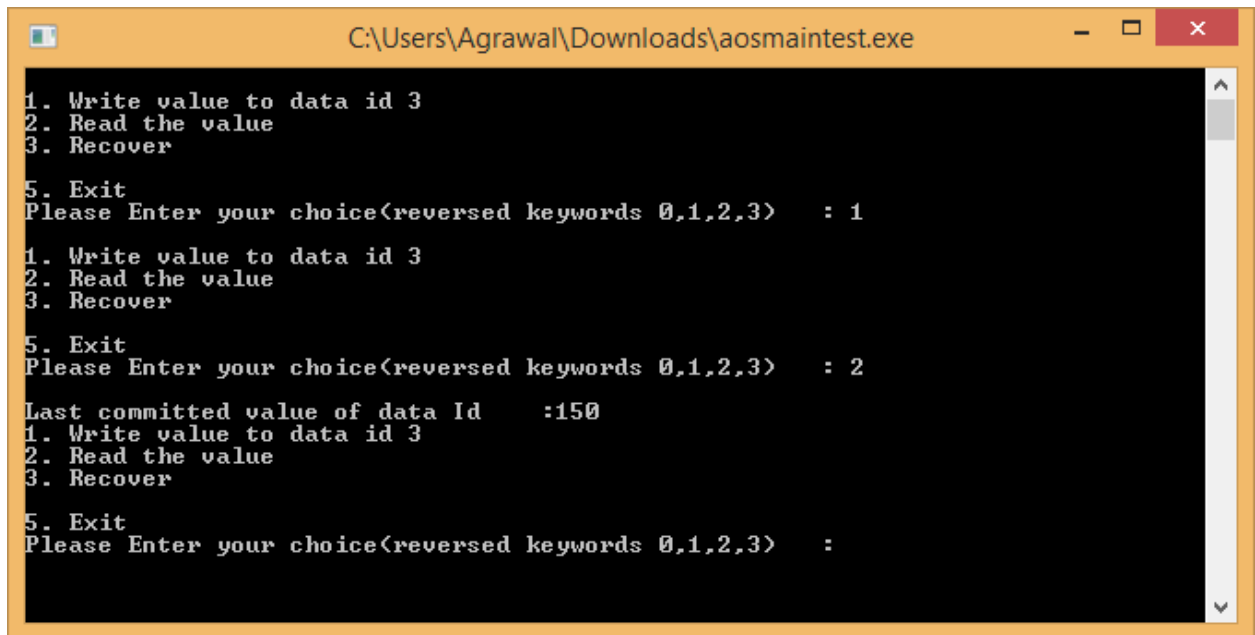
0. Faulty\_Write to a location in disc.
1. Read from that location on disc.
2. Detect (somehow) that the last write was not completed correctly
3. Determine what to do best depending on where the fail occurred.
4. Return either the Old correctly committed value or the new committed value based on step 3.

Expected:

1. Good value is returned and not scrambled information from the faulty\_write.

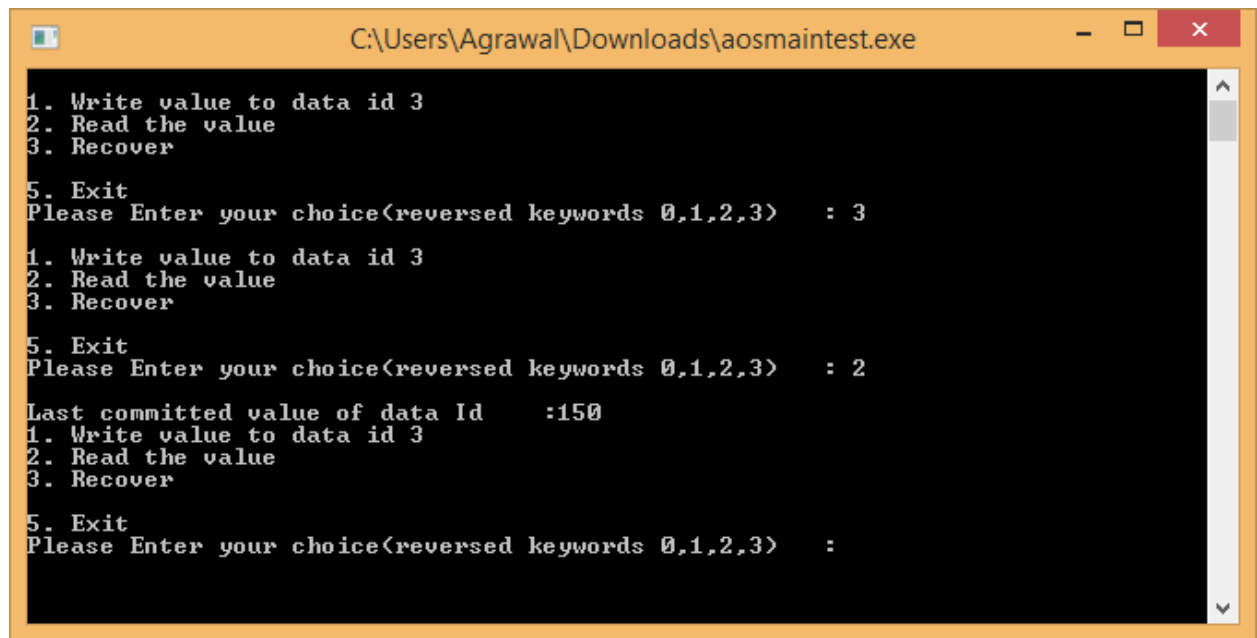
Outcome Snapshot:

- 1) Write a value to data id 3 and reading the value once it is written. As depicted in the screenshot.



```
C:\Users\Agrawal\Downloads\aosmaintest.exe  
1. Write value to data id 3  
2. Read the value  
3. Recover  
5. Exit  
Please Enter your choice(reversed keywords 0,1,2,3) : 1  
1. Write value to data id 3  
2. Read the value  
3. Recover  
5. Exit  
Please Enter your choice(reversed keywords 0,1,2,3) : 2  
Last committed value of data Id :150  
1. Write value to data id 3  
2. Read the value  
3. Recover  
5. Exit  
Please Enter your choice(reversed keywords 0,1,2,3) :
```

- 2) In the shot, We can see after recovery we are fetching the value of data id 3. The output is equal to previous operation that is 150.



```
C:\Users\Agrawal\Downloads\aosmaintest.exe

1. Write value to data id 3
2. Read the value
3. Recover

5. Exit
Please Enter your choice(reversed keywords 0,1,2,3) : 3

1. Write value to data id 3
2. Read the value
3. Recover

5. Exit
Please Enter your choice(reversed keywords 0,1,2,3) : 2

Last committed value of data Id :150
1. Write value to data id 3
2. Read the value
3. Recover

5. Exit
Please Enter your choice(reversed keywords 0,1,2,3) :
```

Outcome:

PASS.

#### ID: Test Case 6

**Purpose: Test Error-Free Environment: Concurrent writing to different blocks with two threads**

Prerequisite:

1. Read, Write, Commit Procedures have been implemented.
2. Error Free read/write procedures

**Command to execute this particular test case:**

- 1) Writing values to different dataid

```
did = 5;
value = 150;
threadid++;
pthread_create(&t[threadid], NULL,&writefunction,NULL);
did = 6;
value = 150;
threadid++;
pthread_create(&t[threadid], NULL,&writefunction,NULL);
```
- 2) Now reading values from the dataids 5 and 6

```
didread = 5;
read_current_value(ha,h,didread);
didread = 6;
```

```
read_current_value(ha,h,didread);
```

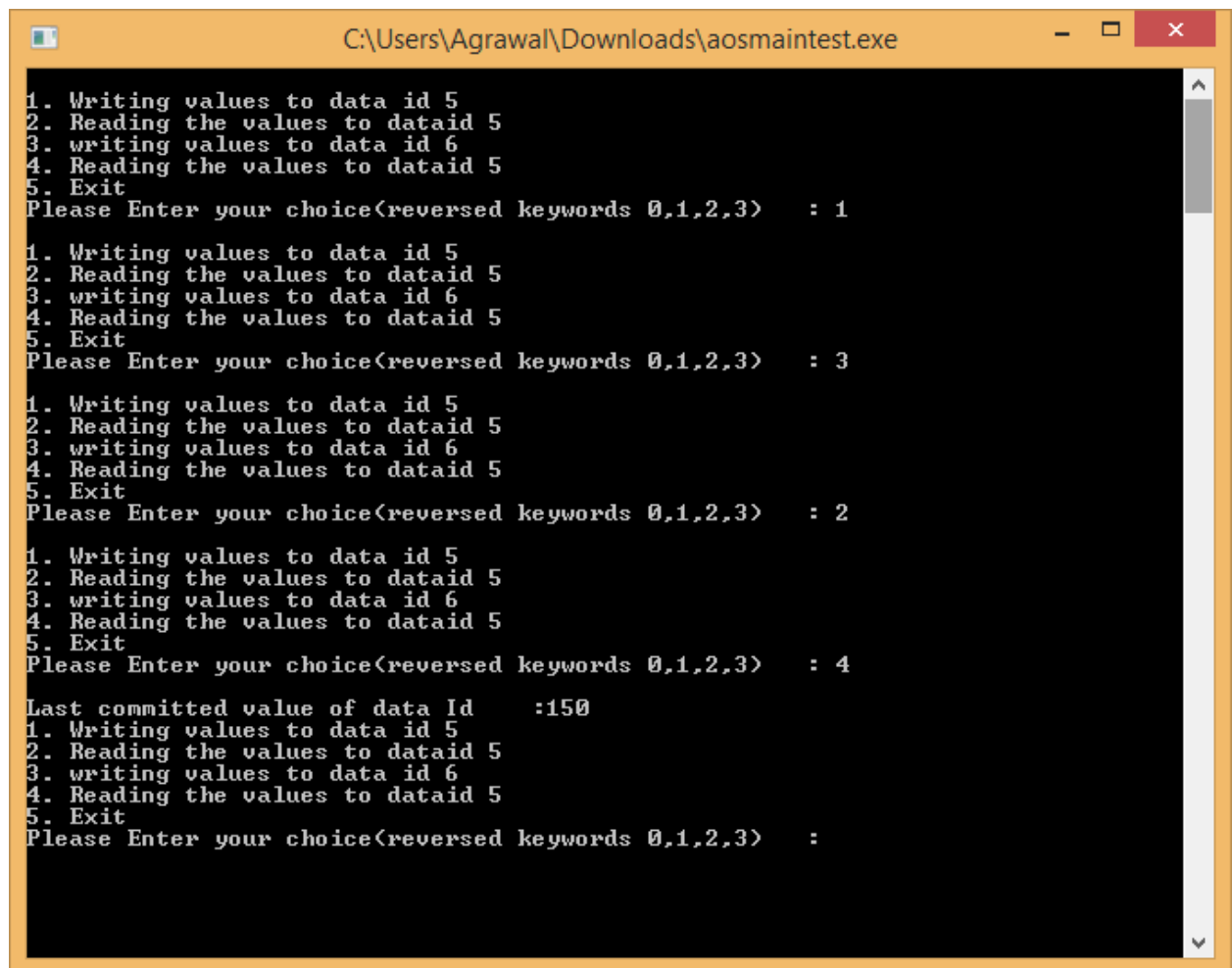
Steps:

0. Faulty Write to a location in disc.
1. Read from that location on disc.
2. Detect (somehow) that the last write was not completed correctly
3. Determine what to do best depending on where the fail occurred.
4. Return either the Old correctly committed value or the new committed value based on step 3.

Expected:

1. Good value is returned and not scrambled information from the faulty write.

Outcome Snapshot:



```
C:\Users\Agrawal\Downloads\aosmaintest.exe

1. Writing values to data id 5
2. Reading the values to dataid 5
3. writing values to data id 6
4. Reading the values to dataid 5
5. Exit
Please Enter your choice(reversed keywords 0,1,2,3) : 1

1. Writing values to data id 5
2. Reading the values to dataid 5
3. writing values to data id 6
4. Reading the values to dataid 5
5. Exit
Please Enter your choice(reversed keywords 0,1,2,3) : 3

1. Writing values to data id 5
2. Reading the values to dataid 5
3. writing values to data id 6
4. Reading the values to dataid 5
5. Exit
Please Enter your choice(reversed keywords 0,1,2,3) : 2

1. Writing values to data id 5
2. Reading the values to dataid 5
3. writing values to data id 6
4. Reading the values to dataid 5
5. Exit
Please Enter your choice(reversed keywords 0,1,2,3) : 4

Last committed value of data Id :150
1. Writing values to data id 5
2. Reading the values to dataid 5
3. writing values to data id 6
4. Reading the values to dataid 5
5. Exit
Please Enter your choice(reversed keywords 0,1,2,3) :
```

As we can see in the snapshot, first we wrote two different values to different data id and then read those values. The output came as 150 and 250.

Outcome:

PASS.

Summary:

In this project, I have implemented Journal Storage System. This system is implemented using outcome records, and data block id. I was able to address the concurrency, recovery and logging mechanism along with maintaining the consistency of the system.

This system also recover all the committed transactions before crash. In future scope, we can address the recovery of data to all transaction just before the crash.

Conclusion:

In conclusion, consistency of the concurrent user system is addressed by locking mechanism. This can be done by recording and validating the current status of the data block id. Sometime, due to unavoidable circumstances system crashes and we lose whole our important data. This system handles the problem mentioned in last statement by using logging and recovery mechanism.

Lessons learned:

- 1) Before and After Atomicity.
- 2) All or nothing atomicity.
- 3) Lock Mechanism.
- 4) Logging
- 5) Recovery
- 6) Concurrency