BTRFS Simulation

Report Content

- What is BTRFS?
- Design of the filesystem simulation and assumptions.
- Implementation - Algorithm
- Instructions to run the program.

What is BTRFS?

BTRFS is an open source filesystem, with copy-on-write (CoW) as the update method. CoW will be discussed in detail in the design part in this document. File system was designed at Oracle Corporation.

Design of the filesystem simulation and assumptions

Programming language used – C++

Used Data structures – Two B trees, Three Sequence containers (vectors) and other primitive data structures like integer, long int ,double etc.

B Trees

- I store text files as strings in a B trees.
- There are Two B trees in my program. I implemented two B trees to simulate the **RAID level 1.** A single B tree in my source code represents a one Hard Disk. I assumed that space allocated to nodes of the B tree number 1 is only taken from Disk 1. I also assumed space allocated for the second B tree 2 is only taken from Disk 2. But actually, all the space is allocated from RAM.
- As we can see in the diagram, The B tree order is 10. A single node can store up to 9 key values. I paired a string variable with a every key value which is called file data. Whenever a file is created, I take a character count of that particular file and divide the file into **8-character sized chunks**.
- So, a single chunk fits into a one file data string. one file can be divided into so many chunks or maybe one chunk. So, this **chunk is actually a block**. A chunk is stored into a single block.
- Only 8 bytes are allocated to the string. So, the block size is 8 Bytes in my virtual file system. Since Btrfs is an extent allocation file system single node of the tree represents a collection of blocks (an extent).

- Altogether B tree node size is 104 Bytes,
  (File data string size) * 9=72 Bytes → i
  All other data (key values, child pointers, leaf bit ... etc.) = 32 Bytes → ii
  So, i + ii = 104 bytes
  Therefore, for an extent there are (104/8) = 13 blocks.

- The key is used to identify a file data block uniquely.
- When file is created it can be allocated to more than one block, so a unique key value is also allocated to every single file data block.

Sequence Containers (Vectors)

The Meta Data List

- Meta data list is used to store the data about files. And it is stored in the RAM.

```
509  struct metaNode //to keep metadata of files
510  {
511      int ID; //starting ID of the file (later this becomes the starting key value for the tree)
512      int numOfBlocks; //number of blocks needed to store the file
513      string name;
514      int size;
515      long int checksum;
516      bool modified_bit;
517
518  };
519
520  static long int id_val=0; //initialize the static variable
521
522  vector<metaNode> metaList;
```

- As we can see in the image, there are 6 members in metadata structure
- **ID** – When a file is created, it can be allocated to more than one block depends on its size, file is divided into 8 byte chunks and those chunks are allocated to *file data blocks*. A unique key value is given to every single *file data block* of the file. From set of file data blocks allocated for the file, the key value of the first block is the ID of that particular file. Because we store a *file data blocks* sequentially.
- id_val variable (in the above image) is used to give key values to these blocks, every time a new *file data block* created id_val is incremented by 1 and, that id_val is assigned to the key of the particular *file data block*.
- **numOfBlocks** – to store the number of blocks needed to store the file.
- **name** – to store the file name
- **checksum** – to store the checksum value of the file. ASCII values of all the characters in the file is added and stored into this variable. When we want to view a file, first we calculate the checksum of file stored on disk1 and disk2, and then compare those values with this original checksum. If it doesn't match, which means file is corrupted. If disk 1's file is corrupted we get the file from disk2 and vice versa. (this simulates a use of **RAID level 1**).
- **Size** – file size is stored in this variable (block size*numOfBlocks)
- **modified_bit –** whether the file is modified or not. (use of this data member will be explained in the next section – CoW lists).

The CoW List Disk1 and The CoW List Disk2

What is copy on write?

Copy on write means, creating a new version of an extent or a page data in a different location. Normally, the data is loaded from disk to memory, modified, and then written elsewhere. The idea is not to update the original location in place, risking a power failure and partial update.

- CoW lists in my program are used to simulate the copy on write feature in Btrfs.

- Whenever a file is modified, it's not going to replace the original content on the B Tree. Just going to store new modified file in the CoW list. And modified bit in the metadata list is set to 1. So, when user wants to access a modified file, it is retrieved from the CoW list. Not the B Tree. But original content is also there in the B tree until user deletes the file or do another modification to the same file.

```
524  struct CoW_list_Node //CoW list (contains the updated files data)
525  {
526      string modifiedFile;
527      int ID;
528      long int checksum;
529  };
530
531  vector<CoW_list_Node> CoW_List_Disk1; // disk 1 CoW list
532  vector<CoW_list_Node> CoW_List_Disk2; // disk 2 CoW list
```

- As we can see, the structure has 3 data members
- **modifiedFile** - stores the modified file content.
- **ID** – file ID (same as before) – so that this is always connected to metadata vector by this.
- **checksum** – the checksum of the modified file
- **Note –** There are two CoW lists (one list per disk). I explained earlier that B Tree 1 represents the disk 1 but actually disk 1 is not only the B Tree 1, disk 1 is represented by the combination of CoW list 1 and B tree 1. I assumed that space allocated to CoW list 1 is allocated from the disk 1 and space allocated to CoW list 2 is allocated from disk 2.

What is RAID Level 1?

RAID 1 is utilized to write and read matching data to pairs of disks. This RAID level is also referred to as data mirroring. Its main function is to allow redundancy that if any of the disks fails, the system will still be able to have data access from other disks.

- To simulate RAID level 1, I used two (B tree + CoW List) structures, altogether two B trees and Two sequence containers.

- Every time a file manipulation (creating, copying, deleting, modifying) happens, it happens in both of the disks.

Why these data structures?

- When a filename is given, my program first searches the metadata list and get the file ID from it. Then search that ID value in the B Tree to get the first block of the file, if file is stored in more than one block, program accesses all those blocks and collect those strings and combine those strings it to get the file.
- I used a B tree to store data because I can retrieve data really fast due its lower height and self-balanced structure. HDD is really slow (access time is higher) compared to RAM so in order retrieve data efficiently we need a data structure like B Tree.
- Hard Disk is a low speed storage compared to the RAM. I used a sequence list to store metadata, because RAM is much faster than HDD and my metadata list is stored in the RAM. So, it doesn't matter that metadata is stored in a list, RAM is fast and I can retrieve data efficiently.
- To store modified files, I used CoW lists because a smaller number of files get modified compared to all the files created in the filesystem. So, it's not so time consuming to access a small list.

Implementation – The Algorithm of the program

Mainly there are 6 functions in the filesystem.

1.createFile() – to create files

2.deleteFile() – to delete files

3.modifyFile() – to modify/update a file

4.listFiles() – to list file names and sizes of all the files in the filesystem

5.viewFile() – to view the content of a file

6.copyFile() – to make a copy of a file

1. createFile()

Step 1. Get the user input
   a. get the file name (should be unique)
   b. get the file content to a string

Step 2. Update metadata vector
   a. calculate checksum, size, number of blocks needed.

Step 3. Store it block by block in **both trees** (RAID level 1)
   a. divide the file content string into 8-character sized chunks
   b. store those chunks one by one and give them IDs (corresponding key values).

2. deleteFile()

step 1. Get the file name from the user

step 2. If file is a modified, then
a. remove from both CoW Lists
step 3. Remove corresponding data from both B trees
step 4. Remove Metadata of the file.

3. modifyFile()

step 1. Get the file name
step 2. Make the file's modified bit 1 in the metadata list
step 3. Display the file to user and gets user input (the modified file)

step 4. Get the modified string and store it to CoW_list vector for both disks

4. listFiles()

loop 1. Iterate through the metadata list items one by one

step 1. if (item's modified bit==1) then

output relevant data from CoW List

continue

step 2. print the relevant data from metadata list

5. viewFile()

step 1. Select the corresponding file that user wants to view
step 2. Get both file strings and calculate their checksums
step 3. IF (file is a modified) then
a. Get a original checksums from a CoW list
b. calculate two saved CoW string's checksums
c. if (original checksum != calculated disk 1's checksum) then
i. if (original checksum != calculated disk 2's checksum)
output "error" (both files are corrupted)
ii. else
Update disk 1's CoW string
Output disk 2 file
d. else
i. if (original checksum != disk 2's checksum)
Update disk 2's CoW string
Output disk 1 file
ii. else
view disk 1 file

ELSE

a. Get the original checksum from the metadata list

b. If (original checksum != calculated disk 1's checksum)
    i. if (original checksum != calculated disk 2's checksum)
        output "error"
    else
        Update disk 1's string
        output disk 2 file
  else
    ii. if (original checksum != calculated disk 2's checksum)
        Update disk 2's string
        view disk 1 file
    else
        view disk 1 file

6. copyFile()

In this function the idea is to give the copy a different name and a different ID but same content.

Step 1. select the correct file

Step 2. give a different name and id to the copy.

Step 3. If (file is a modified one) then

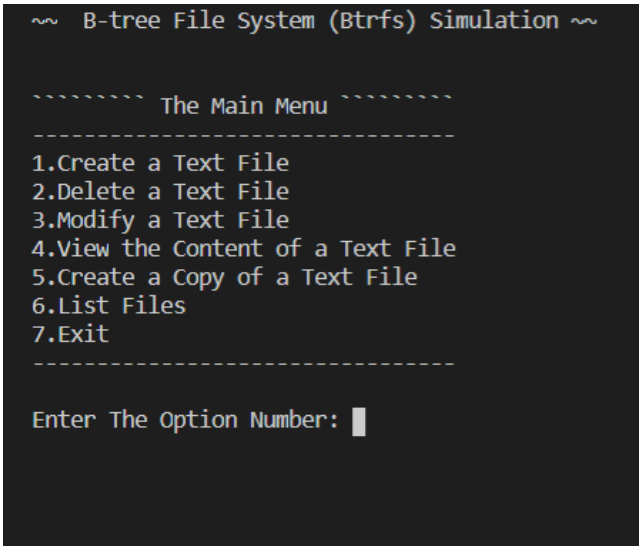
a. get the CoW string and make a copy of it
b. store that new copy to both B Trees (like a new file)
c. update metadata list
(add new copy's information to the metadata list)

else

a. get the file from B Tree and make a copy
b. store that new copy to both B Trees (like a new file)
c. update metadata list
(add new copy's information to the metadata list)

Instruction to run the program

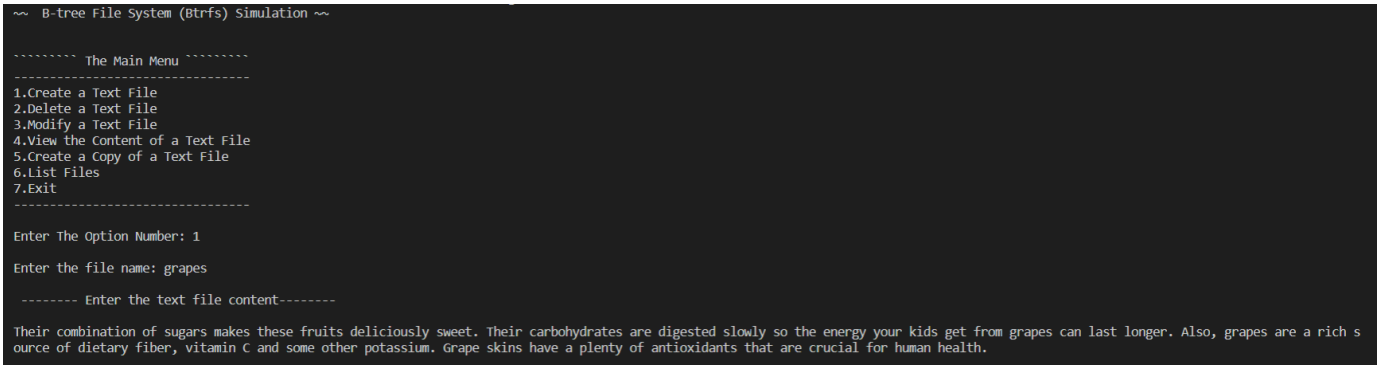
The Main Menu



- This menu will be displayed when we run the program.
- The user can choose any option he/she wants.
- All the basic functionalities are offered in the menu.
- The Menu will be repeatedly displayed until the user decides to exit the filesystem simulation program and enter 7 as the option.

How to Create files?

- Create a text files named grapes.
  - Select the option 1 by entering 1
  - To create a file first enter the name
  - Then enter the content and press enter.



How to View the content of a file?

- View the content of file grapes.
  - Select the option 4
  - Enter the file name

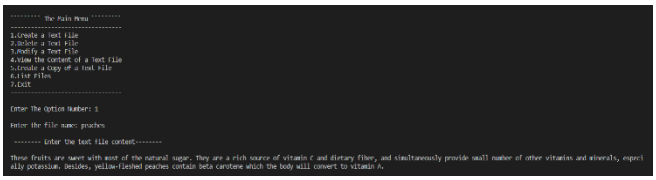
```
`````````` The Main Menu ``````````
---------------------------------
1.Create a Text File
2.Delete a Text File
3.Modify a Text File
4.View the Content of a Text File
5.Create a Copy of a Text File
6.List Files
7.Exit
---------------------------------

Enter The Option Number: 4

Enter the name of the file you want to view: grapes
Text File Content: Their combination of sugars makes these fruits deliciously sweet. Their carbohydrates are digested slowly so the energy your kids get from grapes can last longer. Also,
grapes are a rich source of dietary fiber, vitamin C and some other potassium. Grape skins have a plenty of antioxidants that are crucial for human health.
```

How to delete a text file?

Note - I created two more files called apples and peaches.



- Delete The grapes file.
  - Select the option 2
  - Enter the file name

```
`````````` The Main Menu ``````````
---------------------------------
1.Create a Text File
2.Delete a Text File
3.Modify a Text File
4.View the Content of a Text File
5.Create a Copy of a Text File
6.List Files
7.Exit
---------------------------------

Enter The Option Number: 2

Enter the name of the file you want to delete: grapes
```

How to get the files list?

Note – we deleted the grapes file, so when we view the list, there should be only apples file and peaches file.

- View the list
  - Select the option 6

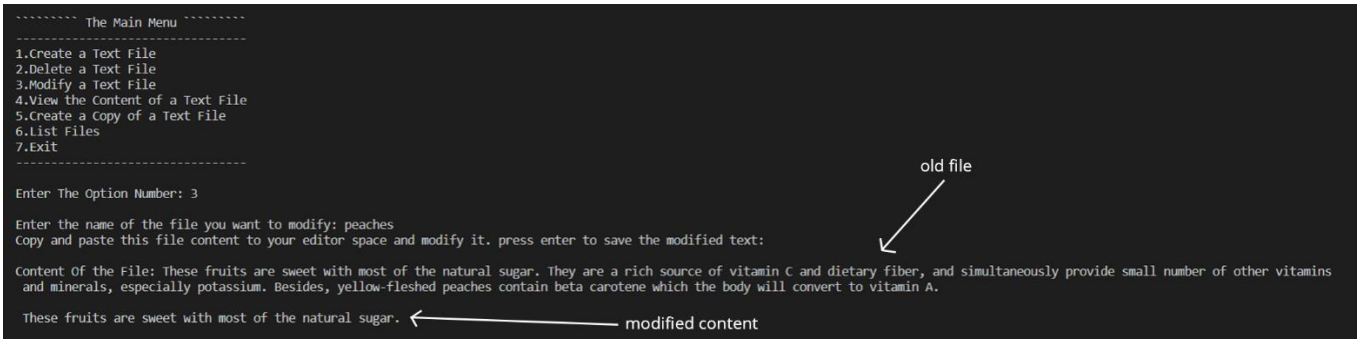
```
`````````` The Main Menu ``````````
---------------------------------
1.Create a Text File
2.Delete a Text File
3.Modify a Text File
4.View the Content of a Text File
5.Create a Copy of a Text File
6.List Files
7.Exit
---------------------------------

Enter The Option Number: 6

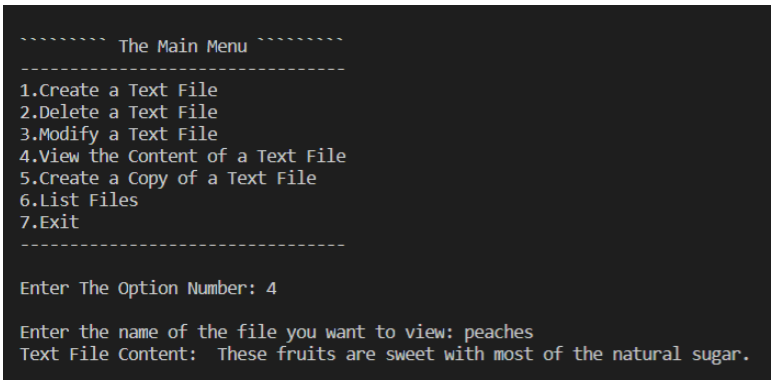
File Name: peaches       |  File Size: 304 Bytes
File Name: apple         |  File Size: 1400 Bytes
```
```

How to modify a file?

- Modify the peaches file
  - Select the option 3
  - Enter the file name
  - Copy paste the current file to the editor
  - Edit the file (Do the modification)
  - Press enter.

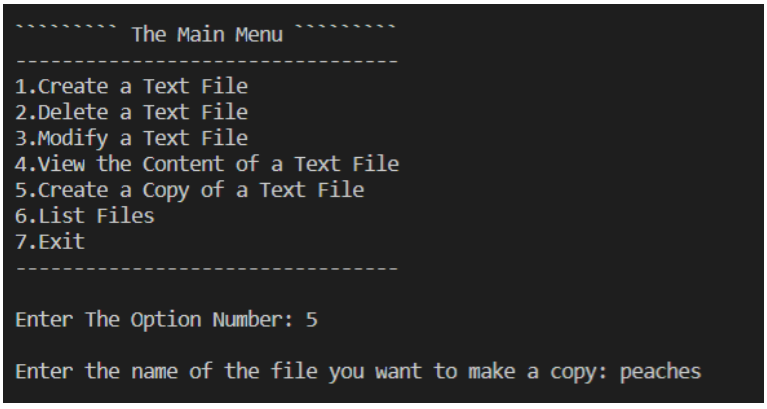


Now, Let's view the content to check whether it is modified.



How to create a copy of a file?

- Create a copy of the peaches file
  - Select the option 5
  - Enter the file name



Now, Let's view the files list to check whether a copy is created.

```
````````` The Main Menu `````````
--------------------------------
1.Create a Text File
2.Delete a Text File
3.Modify a Text File
4.View the Content of a Text File
5.Create a Copy of a Text File
6.List Files
7.Exit
--------------------------------

Enter The Option Number: 6

File Name: peaches       |  File Size: 56 Bytes
File Name: apples        |  File Size: 1400 Bytes
File Name: peaches_copy  |  File Size: 56 Bytes
```
```