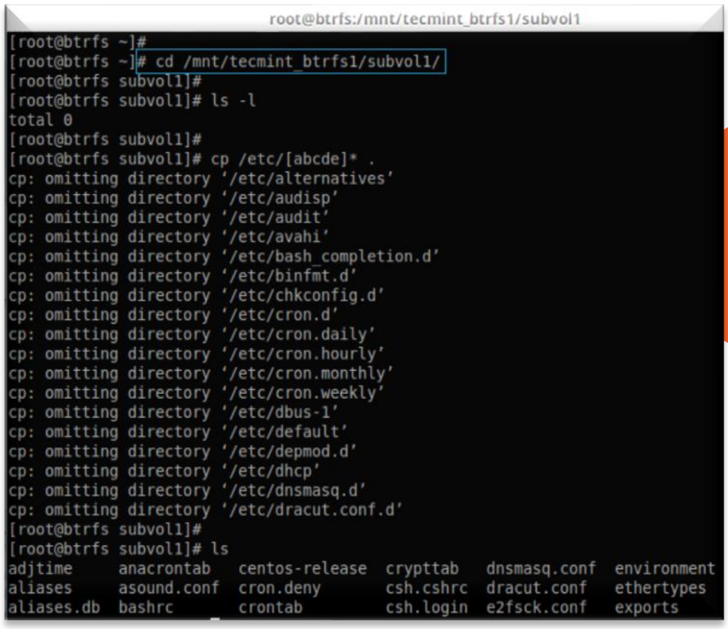


```
root@btrfs:/mnt/tecmint_btrfs1/subvol1
[root@btrfs ~]#
[root@btrfs ~]# cd /mnt/tecmint_btrfs1/subvol1/
[root@btrfs subvol1]#
[root@btrfs subvol1]# ls -l
total 0
[root@btrfs subvol1]#
[root@btrfs subvol1]# cp /etc/[abcde]* .
cp: omitting directory '/etc/alternatives'
cp: omitting directory '/etc/audisp'
cp: omitting directory '/etc/audit'
cp: omitting directory '/etc/avahi'
cp: omitting directory '/etc/bash_completion.d'
cp: omitting directory '/etc/binfmt.d'
cp: omitting directory '/etc/chkconfig.d'
cp: omitting directory '/etc/cron.d'
cp: omitting directory '/etc/cron.daily'
cp: omitting directory '/etc/cron.hourly'
cp: omitting directory '/etc/cron.monthly'
cp: omitting directory '/etc/cron.weekly'
cp: omitting directory '/etc/dbus-1'
cp: omitting directory '/etc/default'
cp: omitting directory '/etc/depmod.d'
cp: omitting directory '/etc/dhcp'
cp: omitting directory '/etc/dnsmasq.d'
cp: omitting directory '/etc/dracut.conf.d'
[root@btrfs subvol1]#
[root@btrfs subvol1]# ls
adjtime     anacrontab   centos-release  crypttab   dnsmasq.conf  environment
aliases     asound.conf  cron.deny       csh.cshrc  dracut.conf   ethertypes
aliases.db  bashrc       crontab         csh.login  e2fsck.conf   exports
```

Btrfs Copy Write File System

Dsa Assignment Take-home 2.0

Name :- W.P Pallewatta

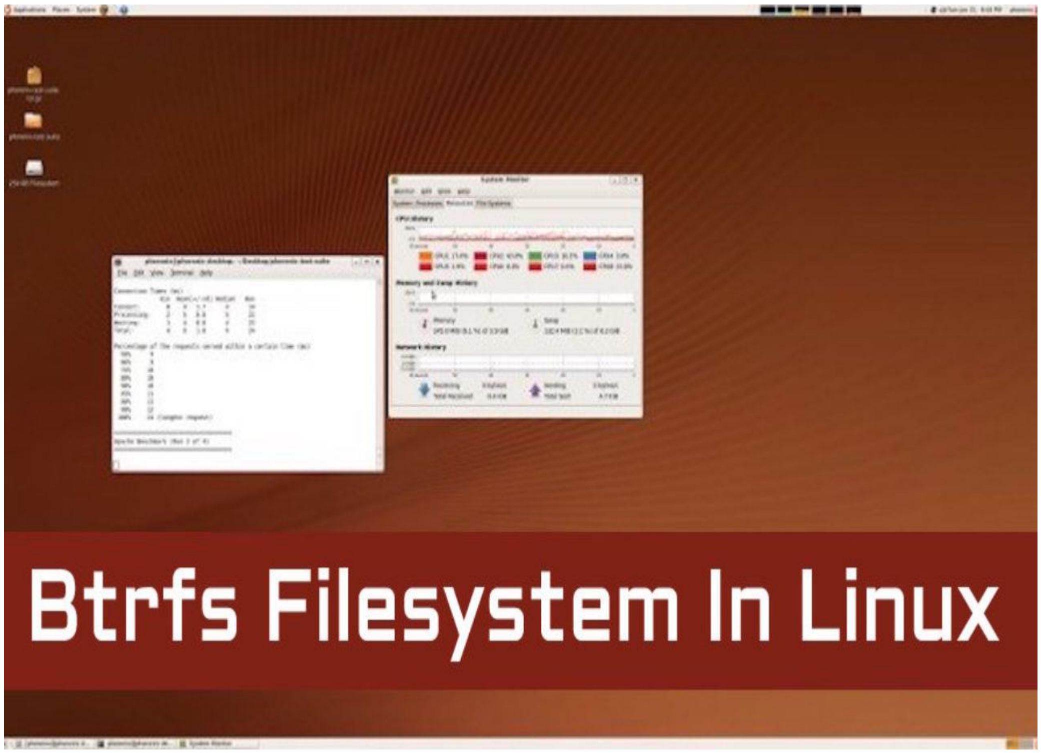
Index No:- 18001149   Topic :- Implementation of Btrfs Copy Write File System Program in C++

Introduction of BTRFS (B Tree File System)

Brief Intro of File Systems Used in Linux

Btrfs, an abbreviation for b-tree file system, is a file system based on the copy-on-write principle, initially designed at Oracle Corporation for use in Linux. The development of Btrfs began in 2007, and since November 2013 the file system's on-disk format has been declared stable in the Linux kernel.

Btrfs is expected to offer better scalability and reliability. It is a copy-on-write file system intended to address various weaknesses in current Linux file systems. Primary focus points include fault tolerance, repair, and easy administration.



Why BTRF File system has been Used for Linux

There are many file systems been used for the store data in a Os's.The Venders used different kind of file systems because of licensing process and also the compatibility.

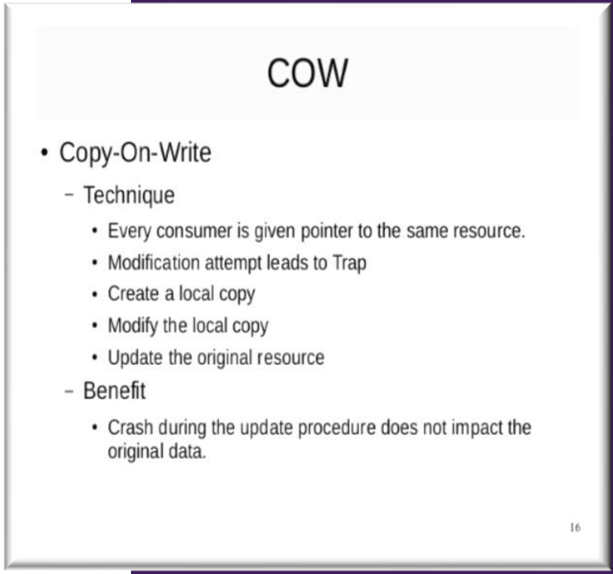
Most of the times Linux are using Btrf file systems because of the accuracy & efficiency.

Why Btrf is Significant from other projects?

Linux's Answers to ZFs and the File & Volume size of 16 EB over multiple devices. Highly integrated into Kernel as Well.

But ZFS aren't Used as a File System in Linux?

- ZFS available as Fuse File system driver
- Sun License issues, internal structures difficult to fit in to the Kernel.
- BTRFS leverages Kernel Code (RAID for Example)



COW

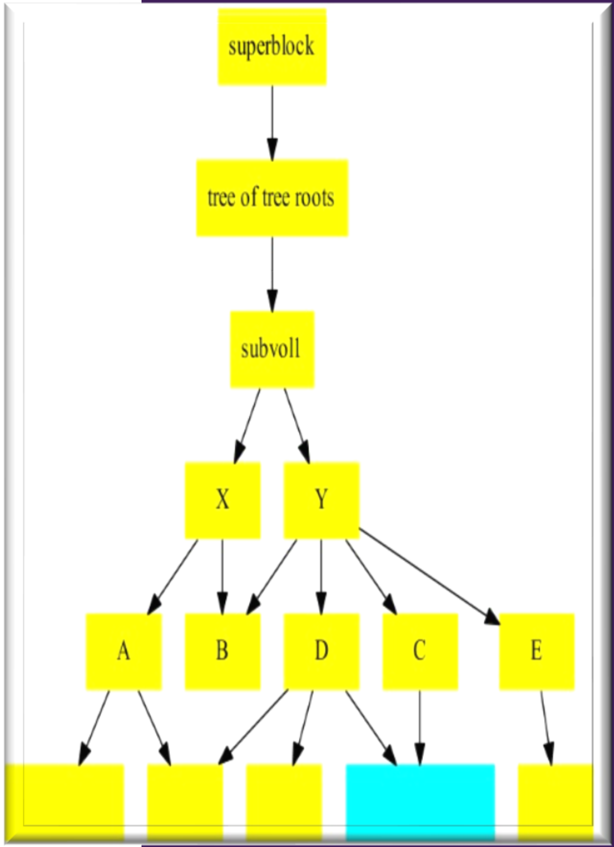
- Copy-On-Write
  - Technique
    - Every consumer is given pointer to the same resource.
    - Modification attempt leads to Trap
    - Create a local copy
    - Modify the local copy
    - Update the original resource
  - Benefit
    - Crash during the update procedure does not impact the original data.

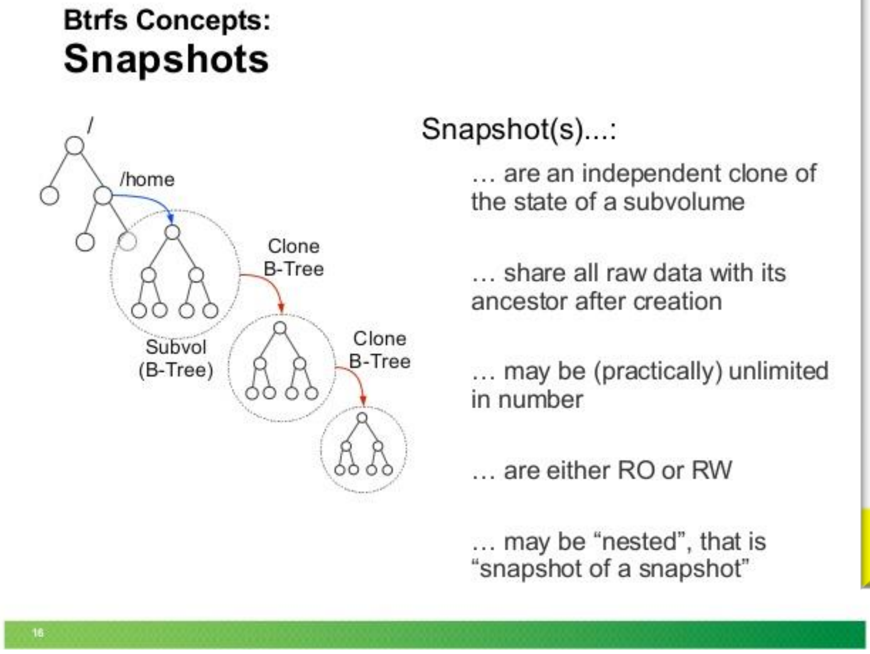
16

BTRFS Overview

- Uses a pool of drivers like ZFS
- Pool divided into sub-volumes
- Copy on Write Philosophy
- B-tree Directory Storage (more Like DB)
- Small Files Stored in Extents.
- SnapShoots:-
  Just Like sub volume Linking to Same data. COW Fashion is being used.

Linux B Tree Filesystem Hierarchy





Btrfs Concepts:
Snapshots

Snapshot(s)...:

... are an independent clone of the state of a subvolume

... share all raw data with its ancestor after creation

... may be (practically) unlimited in number

... are either RO or RW

... may be "nested", that is "snapshot of a snapshot"

16

The Code Implementation of My Btrf File System

1)What is Copy Write Policy?

- Copy only makes another pointer to same data
- Writing one Link leads to real copy
- This works on block level
- Data de-duplication is planned

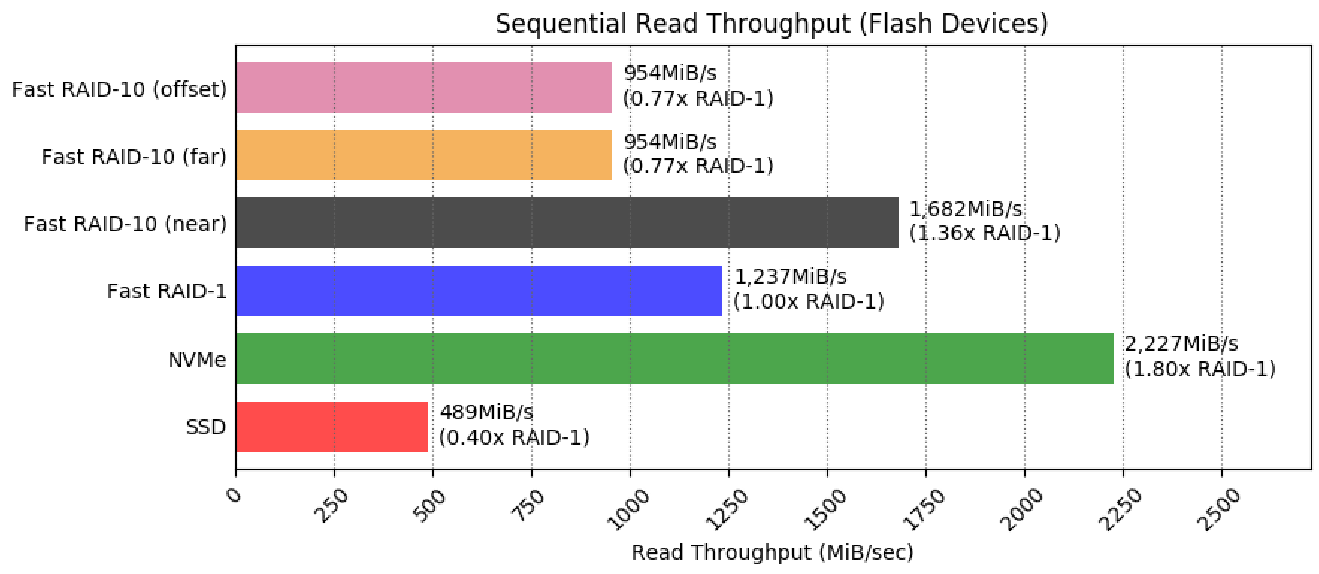
In Linux there are several commands to

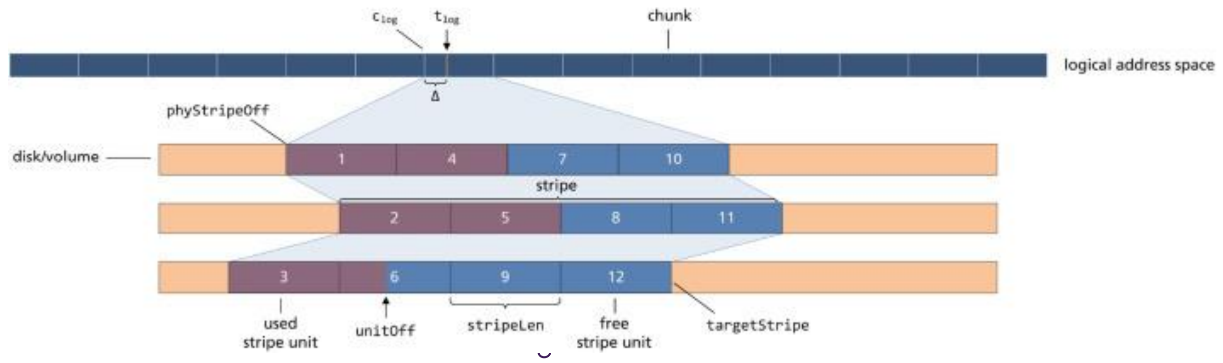
- Making BTRFS File system
- Mount Btrf File System
- Add device, distribute data evenly and remove drive.
- Creating, listing, mounting subvolumes & Taking snapshot

Important Parts in Btrf Linux based File System (Fedora & Nautilus)

- Snapshot taken before update/install
- Any changes can be then roll backed
- Time Line through file change history.

Access time from SSD to Fast Raid



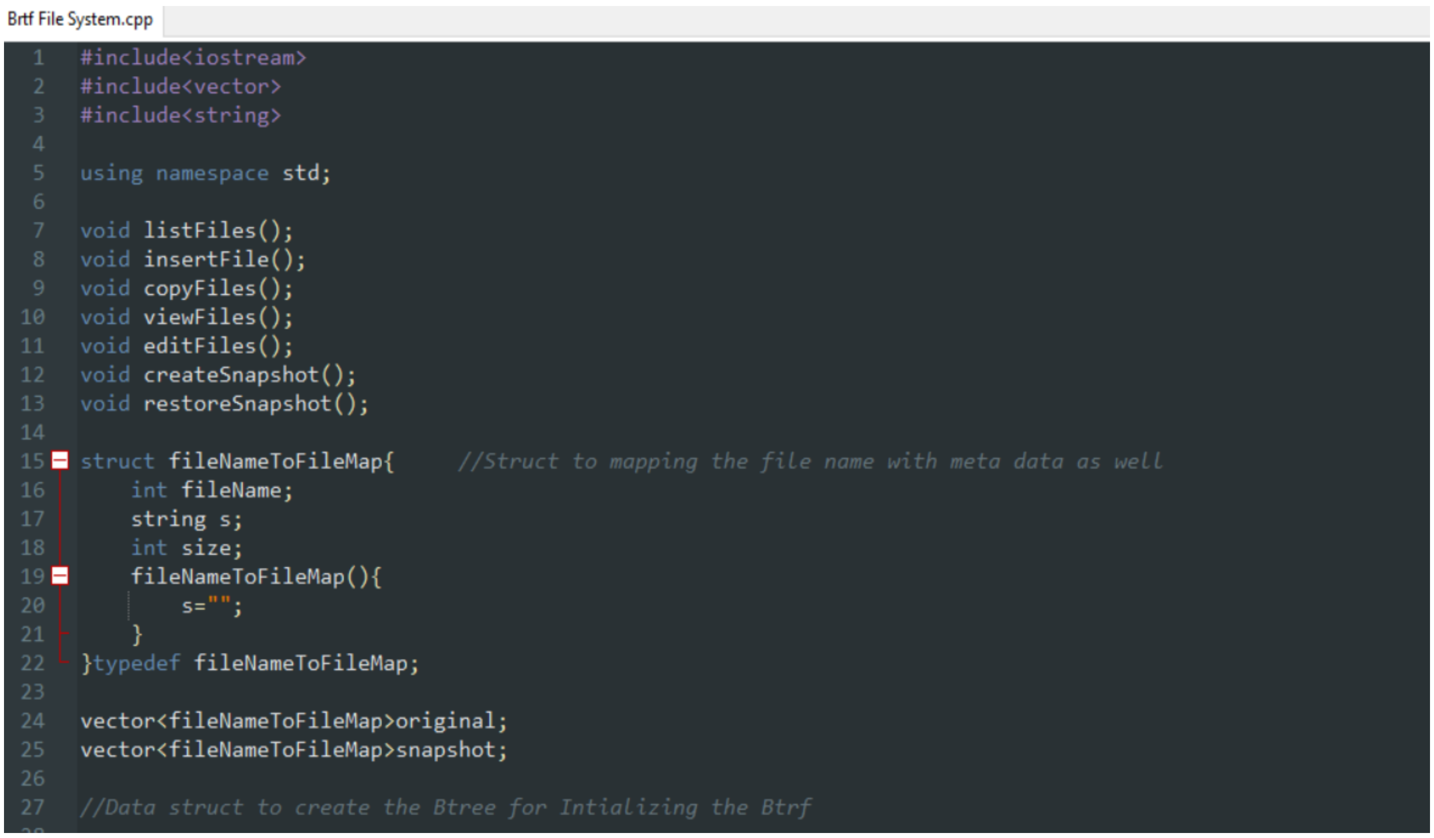


I created/Wrote a program that implemented Using Cpp Object Oriented Knowledge

Implementation of Btree.

For Creating the Btree I used Linked list data structure because Level traversing, searching, copying , Inserting data into linked list is O(1) time complexity. Implementation is little bit difficult but after the data struct is being it is easy to work with the other major Operations.

1)Assigning Functions and Data Structures


Brtf File System.cpp

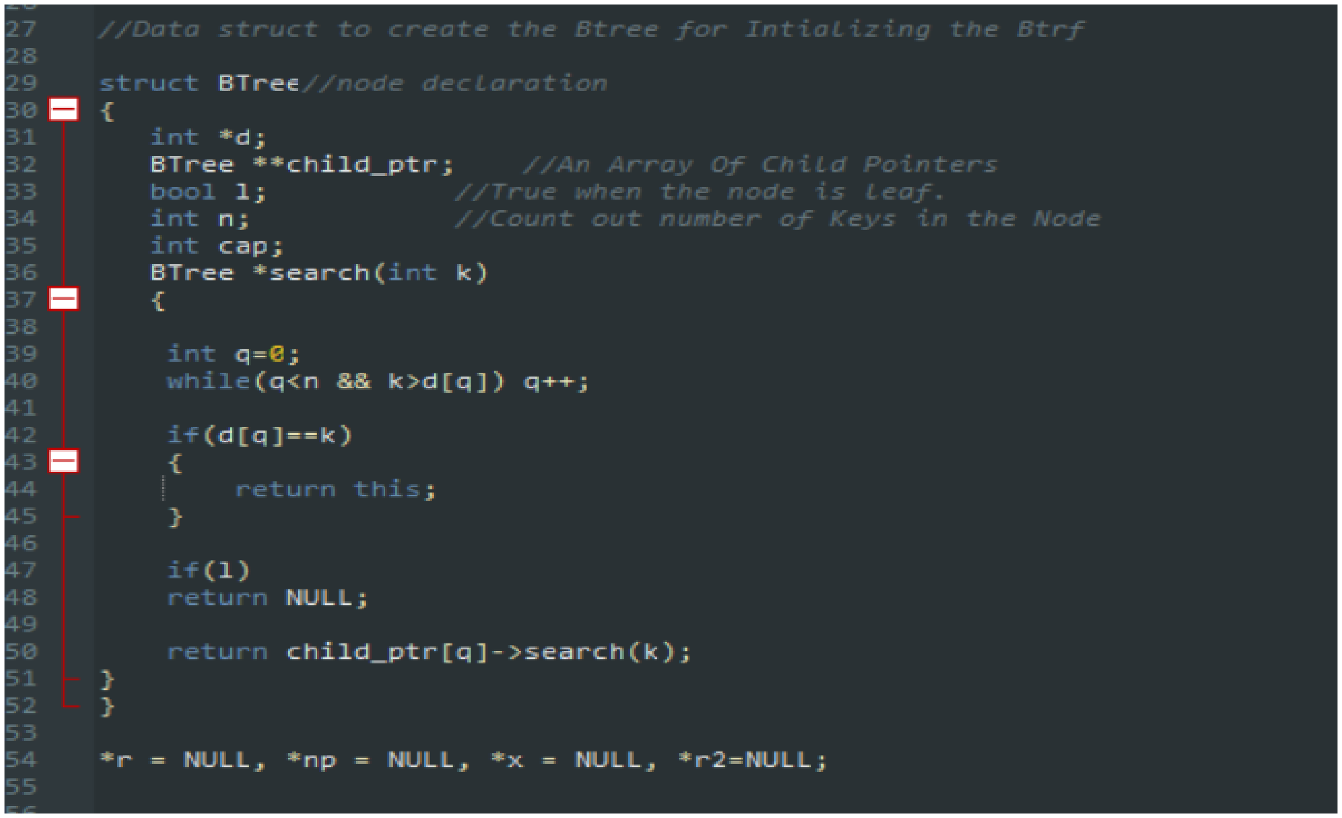
```
#include<iostream>
#include<vector>
#include<string>

using namespace std;

void listFiles();
void insertFile();
void copyFiles();
void viewFiles();
void editFiles();
void createSnapshot();
void restoreSnapshot();

struct fileNameToFileMap{     //Struct to mapping the file name with meta data as well
    int fileName;
    string s;
    int size;
    fileNameToFileMap(){
        s="";
    }
}typedef fileNameToFileMap;

vector<fileNameToFileMap>original;
vector<fileNameToFileMap>snapshot;

//Data struct to create the Btree for Intializing the Btrf
```

In this Snap Shows that my main functions that have been used in My Btrf File System unit. Mainly I have included 7 Functions in my system which can be categorized like

- Listing Files
- Inserting New Files
- Copying One File to Another One
- View Files with the Content
- Edit The file content with in the File
- Creating Snapshot for Clone the Entire File Directory
- Restoring Snapshot

I have also used vectors to dynamically create data structures when I am creating snapshots.

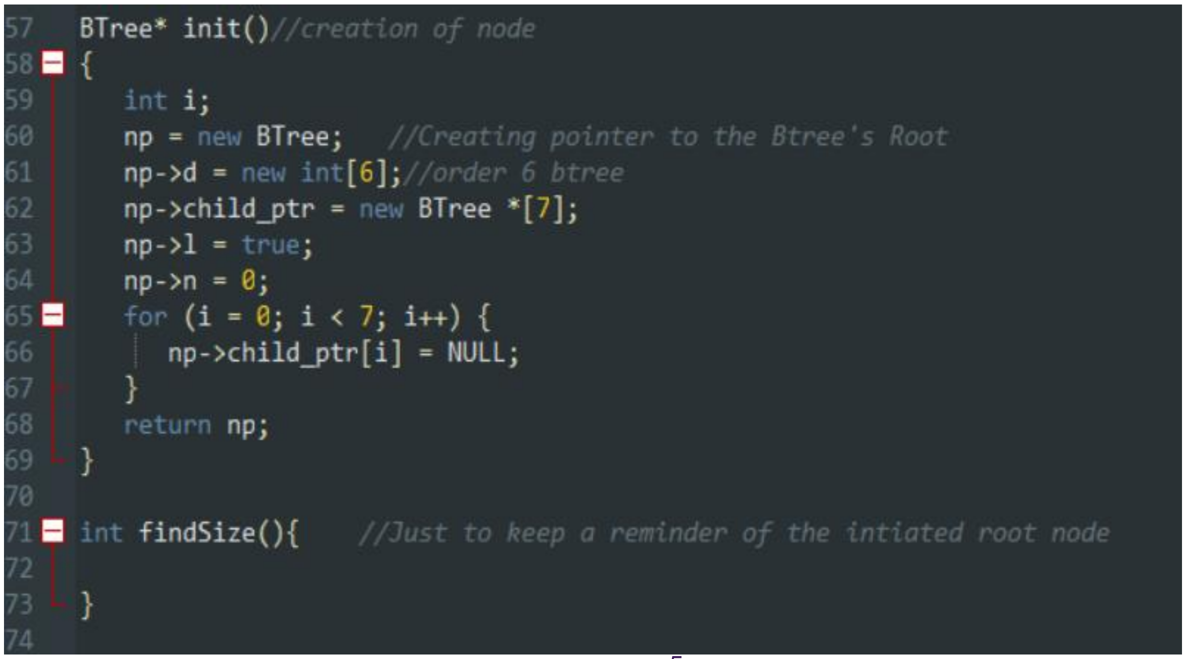
2.Data Declaration for Btree Hierarchy



```
//Data struct to create the Btree for Intializing the Btrf

struct BTree//node declaration
{
    int *d;
    BTree **child_ptr;     //An Array Of Child Pointers
    bool l;               //True when the node is leaf.
    int n;                //Count out number of Keys in the Node
    int cap;
    BTree *search(int k)
    {

     int q=0;
     while(q<n && k>d[q]) q++;

     if(d[q]==k)
     {
         return this;
     }

     if(l)
     return NULL;

     return child_ptr[q]->search(k);
    }
}

*r = NULL, *np = NULL, *x = NULL, *r2=NULL;
```

I decided to use Struct type for Btree in order to data can be easily accessible publicly without having restrictions in class method.

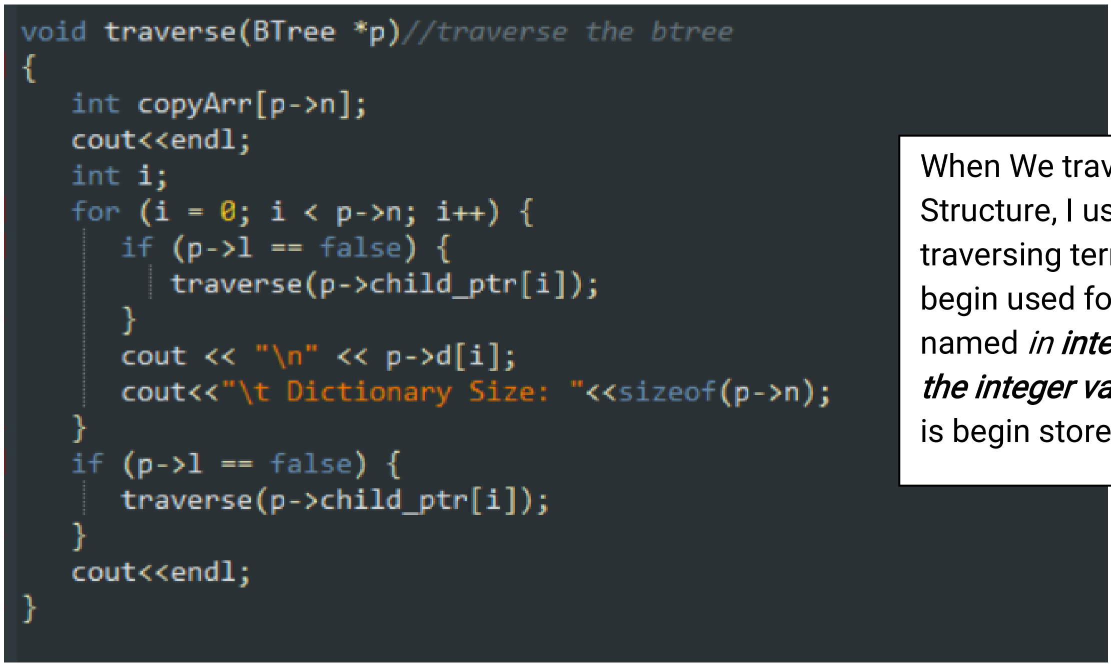
3.Btree Implementation



```
BTree* init()//creation of node
{
    int i;
    np = new BTree;    //Creating pointer to the Btree's Root
    np->d = new int[6];//order 6 btree
    np->child_ptr = new BTree *[7];
    np->l = true;
    np->n = 0;
    for (i = 0; i < 7; i++) {
        np->child_ptr[i] = NULL;
    }
    return np;
}

int findSize(){    //Just to keep a reminder of the intiated root node

}
```

My Intention is to create 7-degree Btree. So 6 keys (Files) can be stored in a Single Node.

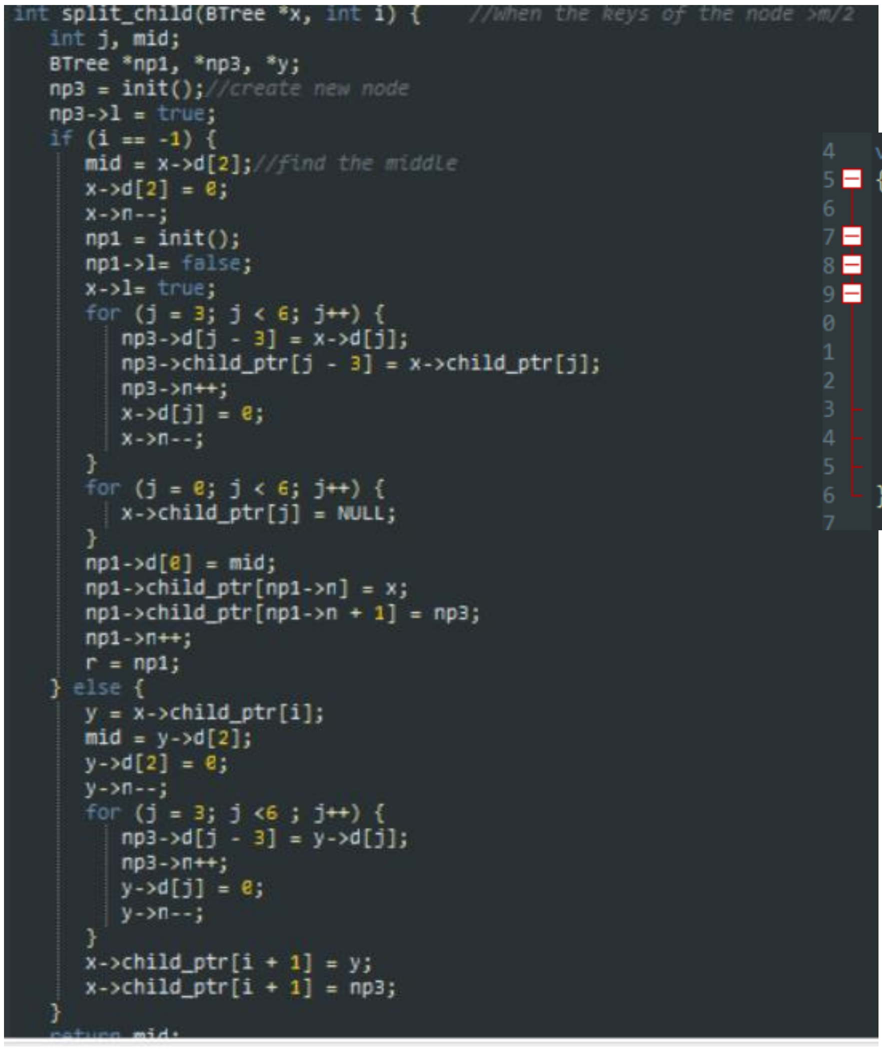
4.Traversing the List Files



```
void traverse(BTree *p)//traverse the btree
{
    int copyArr[p->n];
    cout<<endl;
    int i;
    for (i = 0; i < p->n; i++) {
        if (p->l == false) {
            traverse(p->child_ptr[i]);
        }
        cout << "\n" << p->d[i];
        cout<<"\t Dictionary Size: "<<sizeof(p->n);
    }
    if (p->l == false) {
        traverse(p->child_ptr[i]);
    }
    cout<<endl;
}
```

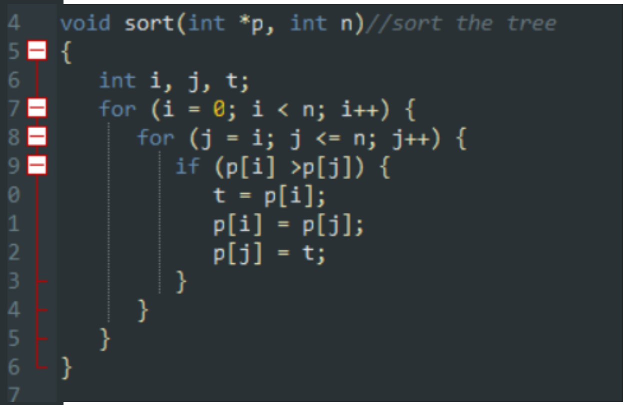
When We traversing through Btree Structure, I used graph theory traversing terminology. Pointers begin used for traversing files are named *in* ***integers, according to the integer value given in the file,*** it is begin stored in the Btree.

5.Spliting the nodes when The Files are Exceeded the Limits



```
int split_child(BTree *x, int i) {    //when the keys of the node >m/2
    int j, mid;
    BTree *np1, *np3, *y;
    np3 = init();//create new node
    np3->l = true;
    if (i == -1) {
        mid = x->d[2];//find the middle
        x->d[2] = 0;
        x->n--;
        np1 = init();
        np1->l= false;
        x->l= true;
        for (j = 3; j < 6; j++) {
            np3->d[j - 3] = x->d[j];
            np3->child_ptr[j - 3] = x->child_ptr[j];
            np3->n++;
            x->d[j] = 0;
            x->n--;
        }
        for (j = 0; j < 6; j++) {
            x->child_ptr[j] = NULL;
        }
        np1->d[0] = mid;
        np1->child_ptr[np1->n] = x;
        np1->child_ptr[np1->n + 1] = np3;
        np1->n++;
        r = np1;
    } else {
        y = x->child_ptr[i];
        mid = y->d[2];
        y->d[2] = 0;
        y->n--;
        for (j = 3; j <6 ; j++) {
            np3->d[j - 3] = y->d[j];
            np3->n++;
            y->d[j] = 0;
            y->n--;
        }
        x->child_ptr[i + 1] = y;
        x->child_ptr[i + 1] = np3;
    }
    return mid;
```

Sorting



```
void sort(int *p, int n)//sort the tree
{
    int i, j, t;
    for (i = 0; i < n; i++) {
        for (j = i; j <= n; j++) {
            if (p[i] >p[j]) {
                t = p[i];
                p[i] = p[j];
                p[j] = t;
            }
        }
    }
}
```

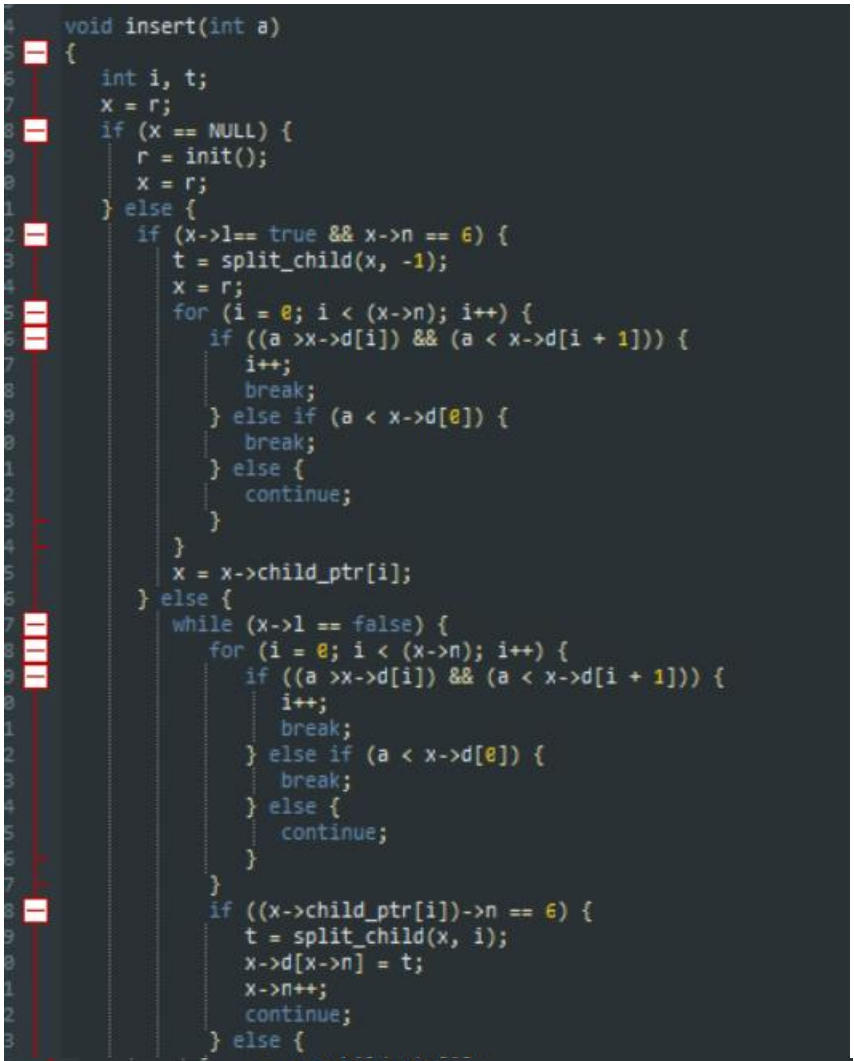
There are 2 Kind of Insertion is begin introduced in My file system which are

***Node isn't Fully occupied**

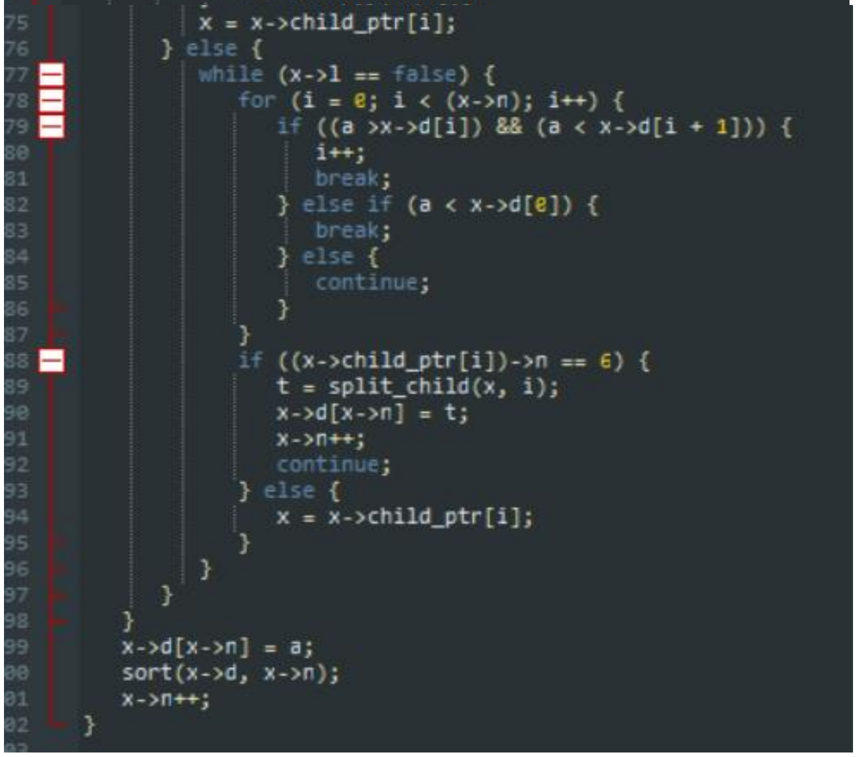
***Node is fully occupied and split operation should be performed**

So on My left side shows that how my first insertion operation will performed.

6.Creating A New File and Store in The Btree



```
void insert(int a)
{
    int i, t;
    x = r;
    if (x == NULL) {
        r = init();
        x = r;
    } else {
        if (x->l== true && x->n == 6) {
            t = split_child(x, -1);
            x = r;
            for (i = 0; i < (x->n); i++) {
                if ((a >x->d[i]) && (a < x->d[i + 1])) {
                    i++;
                    break;
                } else if (a < x->d[0]) {
                    break;
                } else {
                    continue;
                }
            }
            x = x->child_ptr[i];
        } else {
            while (x->l == false) {
                for (i = 0; i < (x->n); i++) {
                    if ((a >x->d[i]) && (a < x->d[i + 1])) {
                        i++;
                        break;
                    } else if (a < x->d[0]) {
                        break;
                    } else {
                        continue;
                    }
                }
                if ((x->child_ptr[i])->n == 6) {
                    t = split_child(x, i);
                    x->d[x->n] = t;
                    x->n++;
                    continue;
                } else {
```



```
                x = x->child_ptr[i];
            } else {
                while (x->l == false) {
                    for (i = 0; i < (x->n); i++) {
                        if ((a >x->d[i]) && (a < x->d[i + 1])) {
                            i++;
                            break;
                        } else if (a < x->d[0]) {
                            break;
                        } else {
                            continue;
                        }
                    }
                    if ((x->child_ptr[i])->n == 6) {
                        t = split_child(x, i);
                        x->d[x->n] = t;
                        x->n++;
                        continue;
                    } else {
                        x = x->child_ptr[i];
                    }
                }
            }
        }
        x->d[x->n] = a;
        sort(x->d, x->n);
        x->n++;
    }
```

MY BTRF'S INSERTION & SPLITING SECTION TO MINIMIZE HEIGHT

In this code segment emphasize that when the inserting is Performed which means that creating a new File with small text residing in it there are several stages should be considered.

1)When a leaf node which consists files isn't fully occupied.

In this code segment preview that when leaf is available file is being is created then stored in the node.

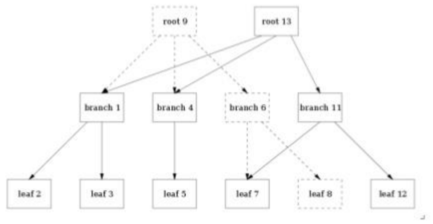
If the root node is fully occupied then splitting have to be performed.

Then the sorting is being done to satisfy the btree properties.

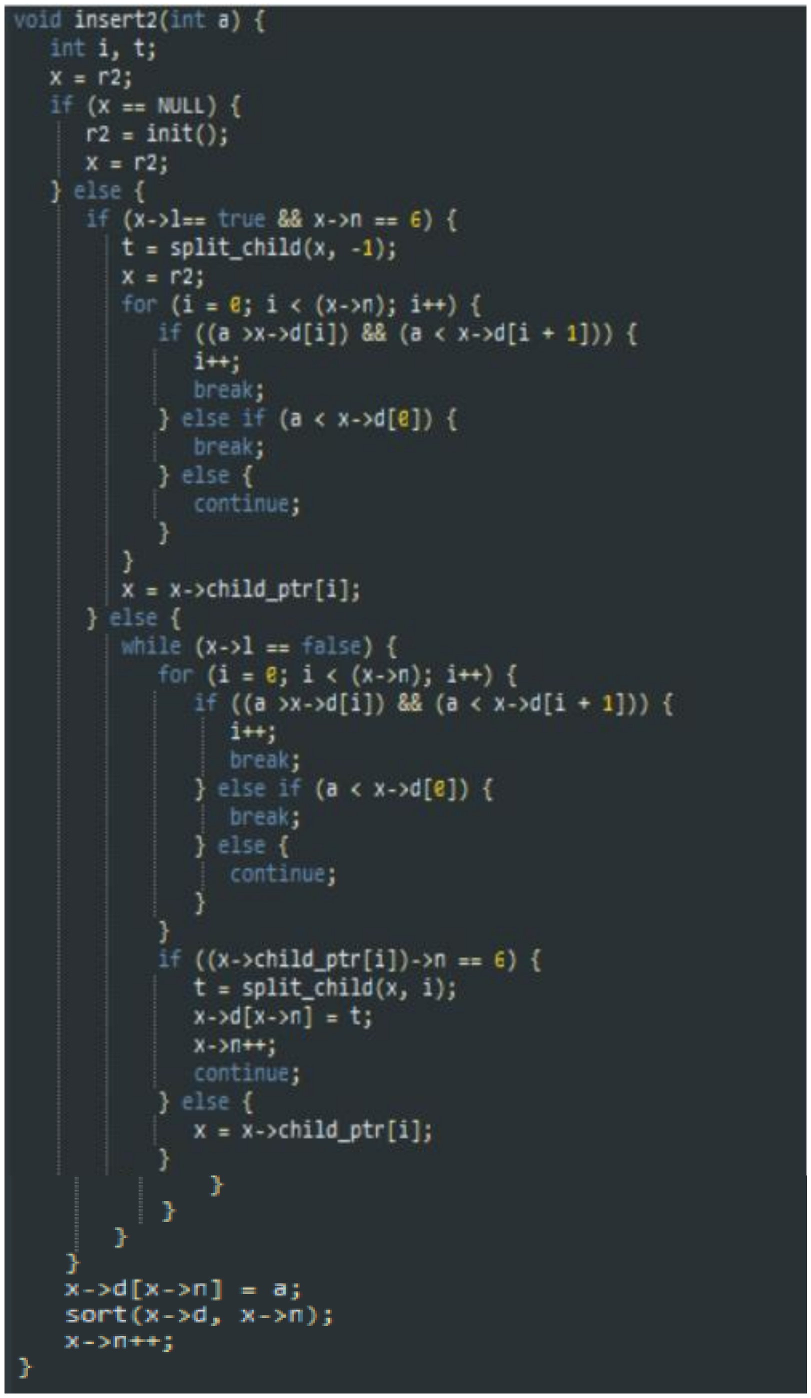
Acknowledgement

BTRFS: Copy on Write

- how the append-only btree works
-



7.When the One Node is Fully Occupied when new File Creation happened



```
void insert2(int a) {
    int i, t;
    x = r2;
    if (x == NULL) {
        r2 = init();
        x = r2;
    } else {
        if (x->l== true && x->n == 6) {
            t = split_child(x, -1);
            x = r2;
            for (i = 0; i < (x->n); i++) {
                if ((a >x->d[i]) && (a < x->d[i + 1])) {
                    i++;
                    break;
                } else if (a < x->d[0]) {
                    break;
                } else {
                    continue;
                }
            }
            x = x->child_ptr[i];
        } else {
            while (x->l == false) {
                for (i = 0; i < (x->n); i++) {
                    if ((a >x->d[i]) && (a < x->d[i + 1])) {
                        i++;
                        break;
                    } else if (a < x->d[0]) {
                        break;
                    } else {
                        continue;
                    }
                }
                if ((x->child_ptr[i])->n == 6) {
                    t = split_child(x, i);
                    x->d[x->n] = t;
                    x->n++;
                    continue;
                } else {
                    x = x->child_ptr[i];
                }
                    }
                }
            }
        }
    x->d[x->n] = a;
    sort(x->d, x->n);
    x->n++;
}
```

WHEN WE CREATING SNAPSHOTS TO CLONE THE BTRFS

This code preview also show that the same process in above process. But I wanted to use two different functions and Insert function is used for creating files & insert2 is being used for creating snap shot of the current btrf.

That's my methodology to reduce the code complexity.

DATA AND METADATA STRUCTURE

•Uses COW (copy on write) tree or Rodeh'sbtrees
Advantages of COW Btrees

➢Increase in the overall depth is infrequent –Minimum number of rearrangement

➢Search could be done in O(log2N)

➢Entire tree need not be in memory

➢COW facility speeds up operations

BTRFS DATA STRUCTURES

•Btrfs internally only knows about three data structures

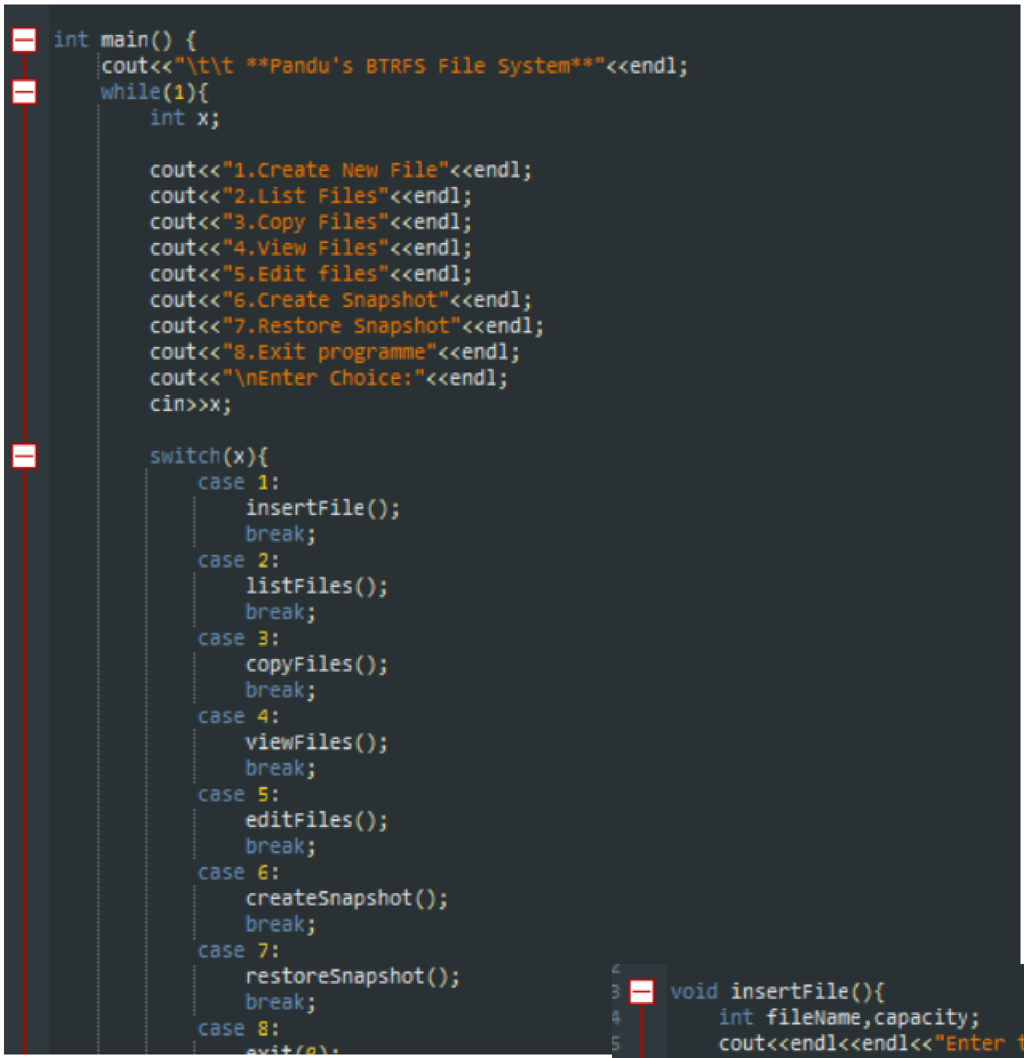
➢Block header -**btrfs_header**

➢Key -**btrfs_key**

➢Item -**btrfs_item**

Btrfs Intro

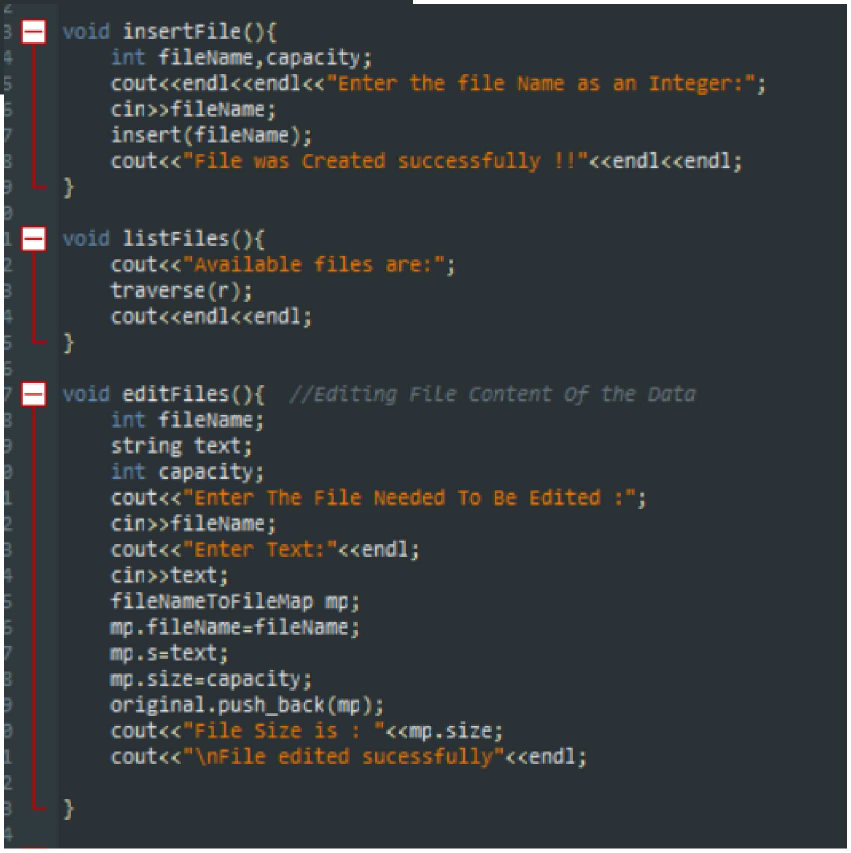
8.Main Function of the My Btrf File System (Main Window)



```
int main() {
    cout<<"\t\t **Pandu's BTRFS File System**"<<endl;
    while(1){
        int x;

        cout<<"1.Create New File"<<endl;
        cout<<"2.List Files"<<endl;
        cout<<"3.Copy Files"<<endl;
        cout<<"4.View Files"<<endl;
        cout<<"5.Edit files"<<endl;
        cout<<"6.Create Snapshot"<<endl;
        cout<<"7.Restore Snapshot"<<endl;
        cout<<"8.Exit programme"<<endl;
        cout<<"\nEnter Choice:"<<endl;
        cin>>x;

        switch(x){
            case 1:
                insertFile();
                break;
            case 2:
                listFiles();
                break;
            case 3:
                copyFiles();
                break;
            case 4:
                viewFiles();
                break;
            case 5:
                editFiles();
                break;
            case 6:
                createSnapshot();
                break;
            case 7:
                restoreSnapshot();
                break;
            case 8:
                exit(0);
```

In this Segment consisting the Interface of the Btrf with the functional requirements one by one.

- Inserting New Files
- Listing Files with Content
- Viewing Content of files
- Edit The File Content & Deleting
- Create a Snapshot using the concept of COW
- Restore The created Snapshot

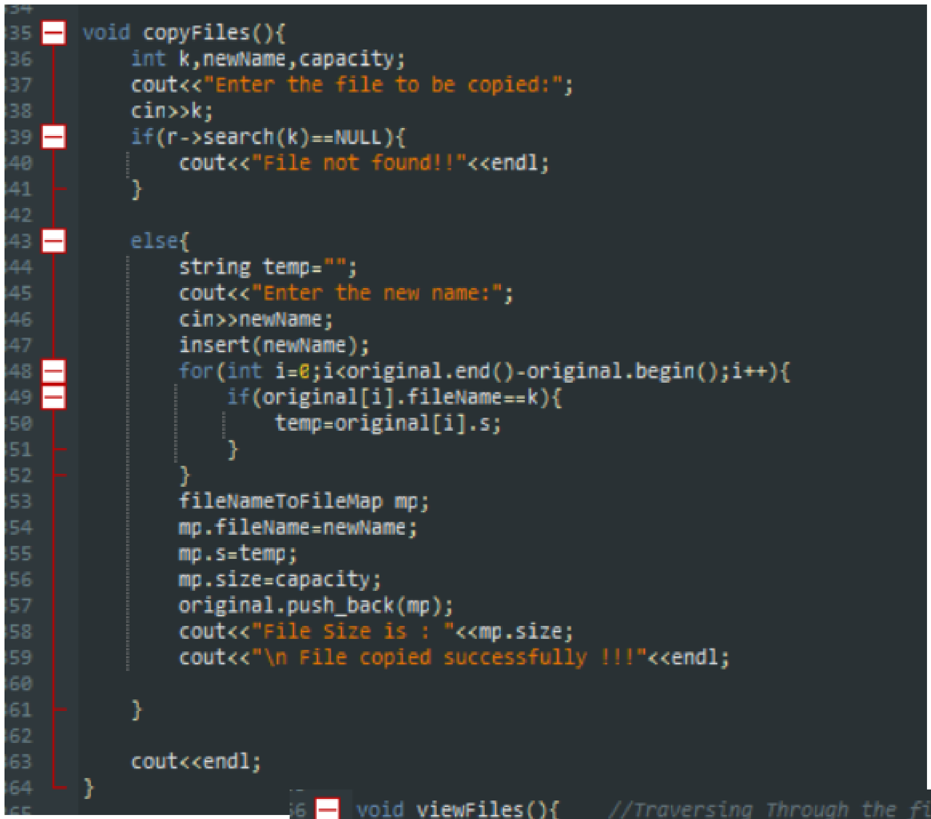


```
void insertFile(){
    int fileName,capacity;
    cout<<endl<<endl<<"Enter the file Name as an Integer:";
    cin>>fileName;
    insert(fileName);
    cout<<"File was Created successfully !!"<<endl<<endl;
}

void listFiles(){
    cout<<"Available files are:";
    traverse(r);
    cout<<endl<<endl;
}

void editFiles(){  //Editing File Content Of the Data
    int fileName;
    string text;
    int capacity;
    cout<<"Enter The File Needed To Be Edited :";
    cin>>fileName;
    cout<<"Enter Text:"<<endl;
    cin>>text;
    fileNameToFileMap mp;
    mp.fileName=fileName;
    mp.s=text;
    mp.size=capacity;
    original.push_back(mp);
    cout<<"File Size is : "<<mp.size;
    cout<<"\nFile edited sucessfully"<<endl;

}
```

FILE SYSTEM TREE

- User-visible files and directories live in file system tree
- Files and directories have a back-reference to parent
- There is one file system tree per sub volume
- Small files are kept in inline extent data item
- Otherwise data is kept outside the tree in extent and extent data item in the tree is used to track them

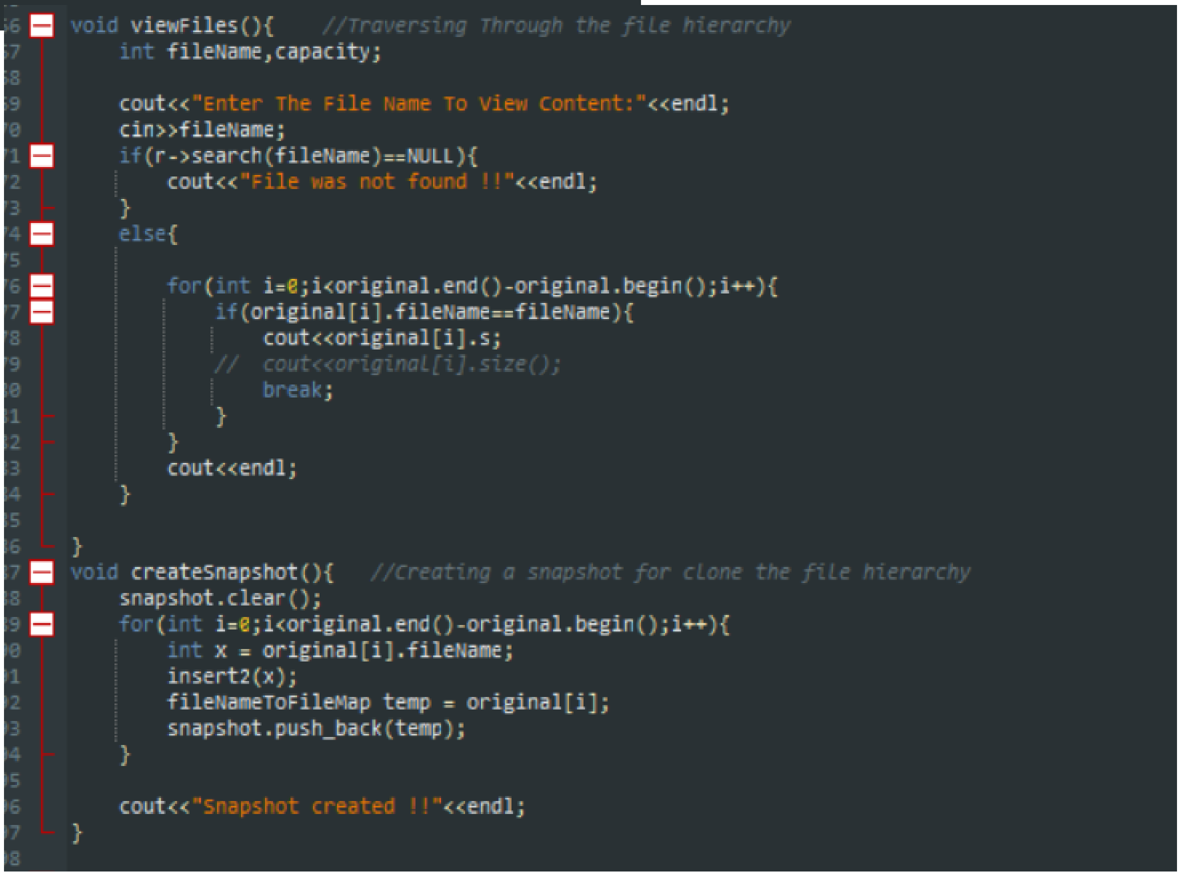
9.File Copy Function and the Viewing Details of the File



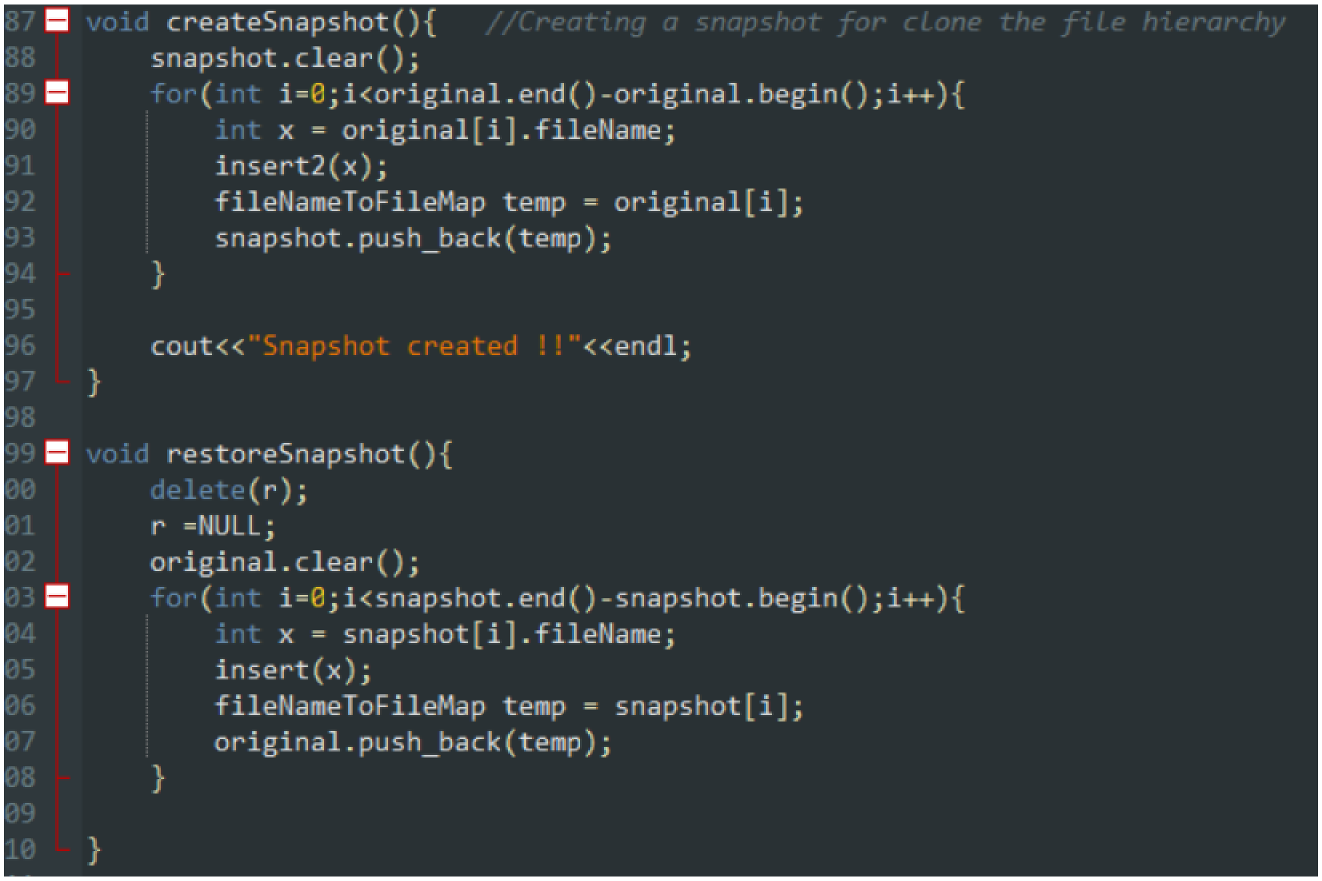
```
void copyFiles(){
    int k,newName,capacity;
    cout<<"Enter the file to be copied:";
    cin>>k;
    if(r->search(k)==NULL){
        cout<<"File not found!!"<<endl;
    }

    else{
        string temp="";
        cout<<"Enter the new name:";
        cin>>newName;
        insert(newName);
        for(int i=0;i<original.end()-original.begin();i++){
            if(original[i].fileName==k){
                temp=original[i].s;
            }
        }
        fileNameToFileMap mp;
        mp.fileName=newName;
        mp.s=temp;
        mp.size=capacity;
        original.push_back(mp);
        cout<<"File Size is : "<<mp.size;
        cout<<"\n File copied successfully !!!"<<endl;

    }

    cout<<endl;
}
```

Key features of btrfs Are

- Dynamic Inode Allocation
- Writable Snapshots And Subvolumes
- Copy On Write
- Compression
- Multiple Device Support
- Space Efficient Packing Of Small Files



```
void viewFiles(){    //Traversing Through the file hierarchy
    int fileName,capacity;

    cout<<"Enter The File Name To View Content:"<<endl;
    cin>>fileName;
    if(r->search(fileName)==NULL){
        cout<<"File was not found !!"<<endl;
    }
    else{

        for(int i=0;i<original.end()-original.begin();i++){
            if(original[i].fileName==fileName){
                cout<<original[i].s;
            //  cout<<original[i].size();
                break;
            }
        }
        cout<<endl;
    }

}
void createSnapshot(){    //Creating a snapshot for clone the file hierarchy
    snapshot.clear();
    for(int i=0;i<original.end()-original.begin();i++){
        int x = original[i].fileName;
        insert2(x);
        fileNameToFileMap temp = original[i];
        snapshot.push_back(temp);
    }

    cout<<"Snapshot created !!"<<endl;
}
```



```
void createSnapshot(){    //Creating a snapshot for clone the file hierarchy
    snapshot.clear();
    for(int i=0;i<original.end()-original.begin();i++){
        int x = original[i].fileName;
        insert2(x);
        fileNameToFileMap temp = original[i];
        snapshot.push_back(temp);
    }

    cout<<"Snapshot created !!"<<endl;
}

void restoreSnapshot(){
    delete(r);
    r =NULL;
    original.clear();
    for(int i=0;i<snapshot.end()-snapshot.begin();i++){
        int x = snapshot[i].fileName;
        insert(x);
        fileNameToFileMap temp = snapshot[i];
        original.push_back(temp);
    }

}
```

WRITABLE SNAPSHOT AND SUBVOLUME

•Allows creation of writable snapshots

•They can be used for backups and testing transactions

•Occupied space actually increases only when data is written to snapshot

•Sub volumes are named btrees

•They have inodes inside the tree of tree roots

•Snapshots are sub volumes whose root is shared with another

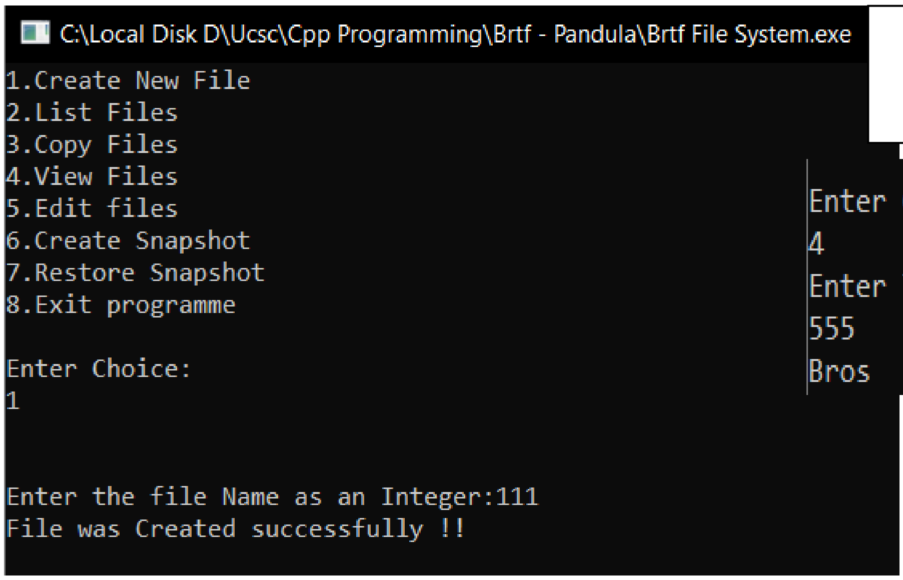
•They can be mounted as unique root to prevent access

In this code segment I have used vectors to dynamically allocate data structure when the create snapshot function is used. Also when a file is being misplaced it that can be restored by using restore snapshot function. So the Basic Operations of my btrfs file system is initiated like this. Mainly Creating a Cloning file system using a snapshot which create raid level 1.So we can restore data from the clone.

My Btrf File System (Program Execution)

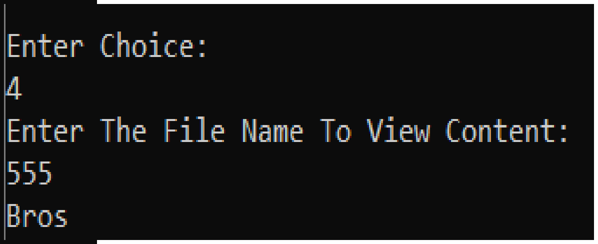
Assumptions

- My files are named In Integers
- Files begin stored according to the ascending order of files.
- Dictionaries and the file sizes are given in Bytes.

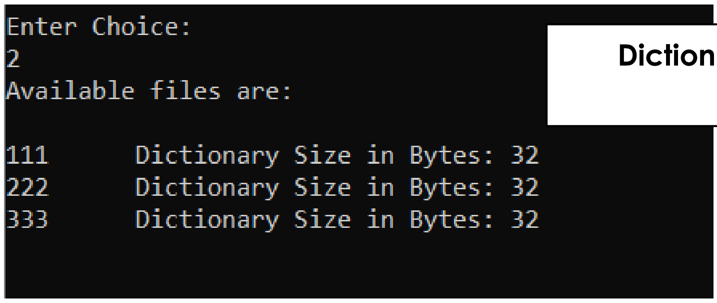


```
C:\Local Disk D\Ucsc\Cpp Programming\Brtf - Pandula\Brtf File System.exe
1.Create New File
2.List Files
3.Copy Files
4.View Files
5.Edit files
6.Create Snapshot
7.Restore Snapshot
8.Exit programme

Enter Choice:
1


Enter the file Name as an Integer:111
File was Created successfully !!
```

Creating Files In the Btrf and Viewing the Small text file Contents

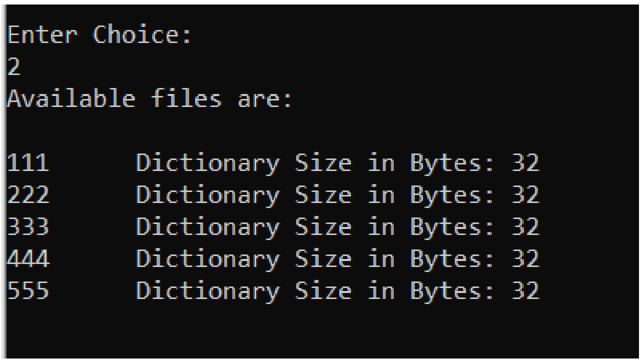


```
Enter Choice:
4
Enter The File Name To View Content:
555
Bros
```

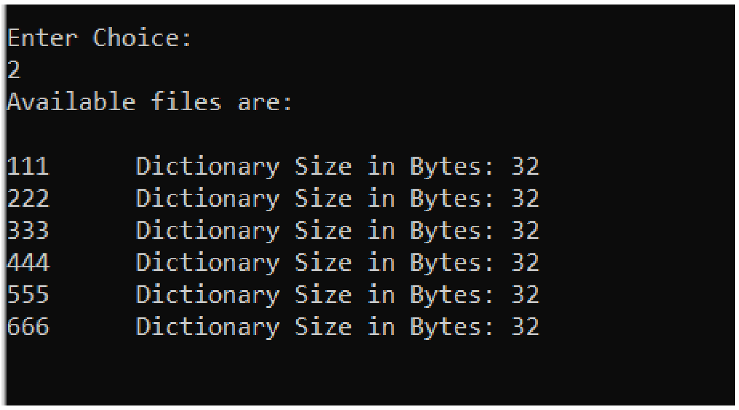


```
Enter Choice:
2
Available files are:

111      Dictionary Size in Bytes: 32
222      Dictionary Size in Bytes: 32
333      Dictionary Size in Bytes: 32
```

Dictionary View of the File System

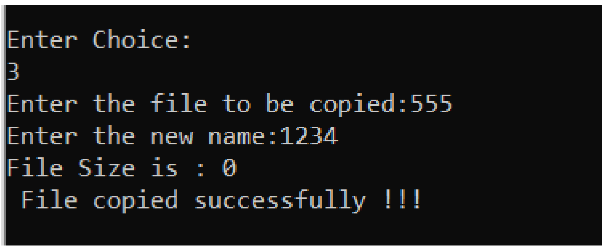


```
Enter Choice:
2
Available files are:

111      Dictionary Size in Bytes: 32
222      Dictionary Size in Bytes: 32
333      Dictionary Size in Bytes: 32
444      Dictionary Size in Bytes: 32
555      Dictionary Size in Bytes: 32
```



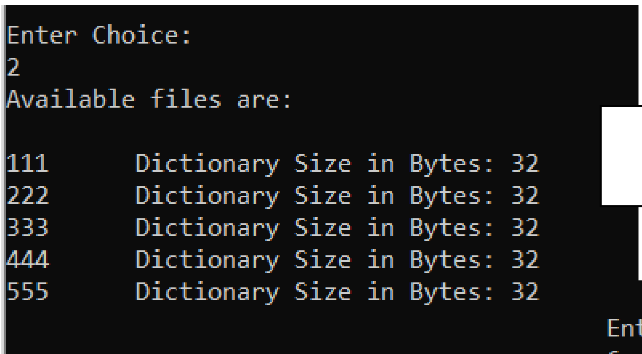
```
Enter Choice:
2
Available files are:

111      Dictionary Size in Bytes: 32
222      Dictionary Size in Bytes: 32
333      Dictionary Size in Bytes: 32
444      Dictionary Size in Bytes: 32
555      Dictionary Size in Bytes: 32
666      Dictionary Size in Bytes: 32
```

Before Deletion of Dictionary View & After Deletion of the File System



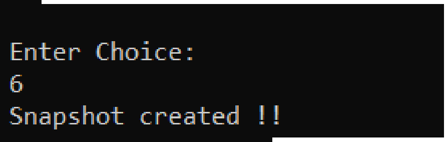
```
Enter Choice:
3
Enter the file to be copied:555
Enter the new name:1234
File Size is : 0
 File copied successfully !!!
```

Coping One Directory and Renaming With another File Type but the content is Same



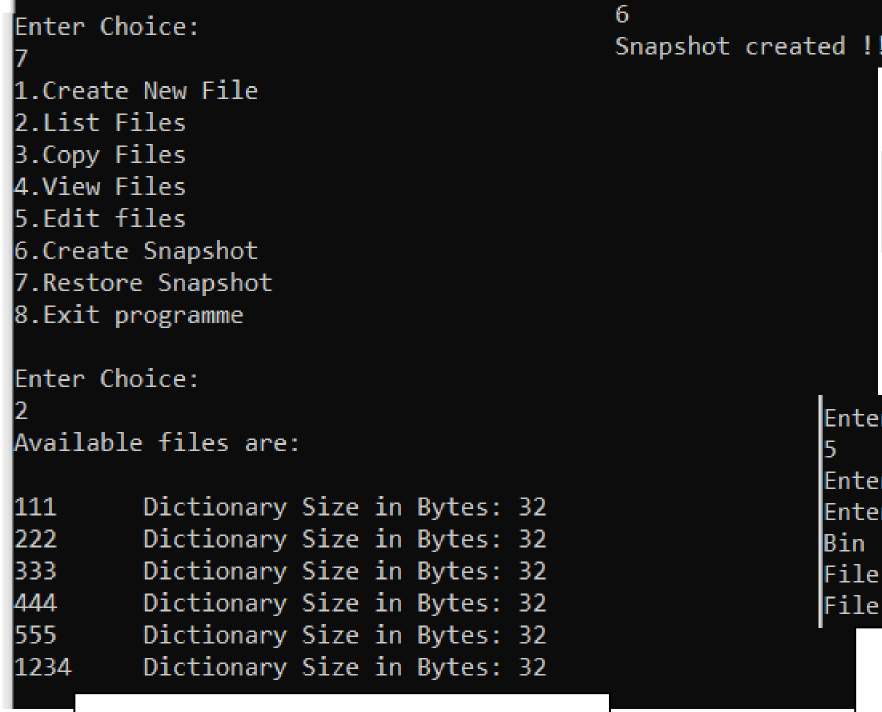
```
Enter Choice:
2
Available files are:

111      Dictionary Size in Bytes: 32
222      Dictionary Size in Bytes: 32
333      Dictionary Size in Bytes: 32
444      Dictionary Size in Bytes: 32
555      Dictionary Size in Bytes: 32
```

Before the Snapshot is taken File hierarchy



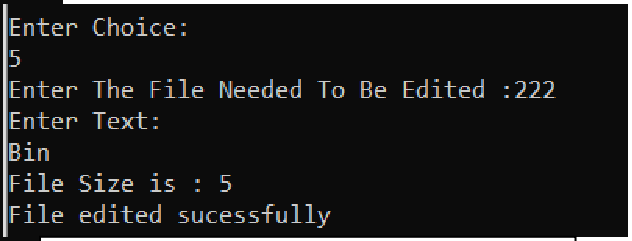
```
Enter Choice:
6
Snapshot created !!
```

Snapshot is being created for the protection of Data



```
Enter Choice:
7
1.Create New File
2.List Files
3.Copy Files
4.View Files
5.Edit files
6.Create Snapshot
7.Restore Snapshot
8.Exit programme

Enter Choice:
2
Available files are:

111      Dictionary Size in Bytes: 32
222      Dictionary Size in Bytes: 32
333      Dictionary Size in Bytes: 32
444      Dictionary Size in Bytes: 32
555      Dictionary Size in Bytes: 32
1234     Dictionary Size in Bytes: 32
```

After the Snapshot Restoration is Completed



```
Enter Choice:
5
Enter The File Needed To Be Edited :222
Enter Text:
Bin
File Size is : 5
File edited sucessfully
```

Text File Can be Created to store in the Btrf

So Using Btree data structure We can Implement Btrf File System using Cow method to Easily access the data with in a Finite Number Of Seconds. So this File system is mainly used In Linux Kernel.

W.P Pallewatta - 18001149