Project:

Inter Planetary System

Discription:

I order to implement the “Inter Planetary System” as a project, we implemented different parts of program into different files. Let’s discuss it file wise.

1- Main

Here in this file we included all other files. Starting the program the program asks for the **identifier space** from the user. It is set accordingly by the user. The program asks for the **Machine’s Id**, whether the program automatically assigns the new machine or the user explicitly assigns it, like wise it also takes user inputs for **Order of the tree** and **Number of the total machines**. This forms a collection of machines in an environment. The name of each machine is entered by the user. A key is automatically generated as the machine is added. Furthermore the given functionalities are offered to the user:

(1) Add a Data in form of key, value
(2) Remove a Data by using key
(3) Print the Routing Table
(4) Print the B-Tree
(5) Add new Machines without disrupting the Functionality
(6) Delete any Machine without disrupting the Functionality of DHT
(7) Display Ring

2- RingDHT

In this file **class: Routing Node** is made which has pointer to next and previous. Furthermore it also has variable **Node** of type **Machine**.

Class: Machine contains **pointer to B Tree. Machine Id, next** pointer, **machine’s name, ispace** contains the identifier space in the form of integer. **bitSpace** contains **identifier space** converted to hexadecimal. It also contains a pointer to **Finger Table**

Whenever a new machine is created, a B-tree as well as a finger table is created for each machine. This class also include functions which are discussed below:

1- **Successor Function**

   It is finding the successor of it when used in Finger Table

2- **makeFT**

   It involves iterating through the Finger Table entries and assigning values to them. For each entry, it calculates a key based on the node's identifier and Id, it adjusts the key to fall within a specified range, and then assigns the successor node. This process is repeated for each entry in the Finger Table.

3- **Display FT**

   It just displays the made Finger table.

4- **Insert File**

   This function is designed to insert a new file into the appropriate location in a distributed file system. It begins by printing information about the current machine and then traverses the ring DHT to determine the correct node to handle the file insertion. If the current machine's identifier matches the file's key, it inserts the file into its B-tree. Otherwise, it navigates through the ring using the Finger Table, identifying the successor node for the file's key and recursively calls the function on that node until the file is successfully inserted.

5- **Delete File**

   This function is designed for deleting a file with the specified key in a distributed file system. It starts by printing the information about the current machine and then traverses the ring DHT to locate the correct node for the file deletion. If the current machine's identifier matches the file's key, it deletes the key from its B-tree. Otherwise, it navigates through the ring using the Finger Table. After finding the successor node for the key, and recursively calls function on that node until the file is successfully deleted.

6- **Find B-Tree**

   This function searches for a B-tree associated with a specific machine identifier in a distributed file system. If the current machine's identifier matches the target identifier, it prints the B-tree's contents in an in-order traversal. Otherwise, it navigates through the ring using the Finger Table. It finds the successor node for the target identifier, and recursively calls the function on that node until the associated B-tree is found and displayed.

<u>Class: Ring DHT</u> represents a Chord-based distributed hash table (DHT) implemented as a singly circular linked list.
It contains following functions:

(1) Constructor
Initializes the DHT with parameters such as bit space, tree order, uniqueness criteria, and whether the identifiers are set manually.

(2) Update Routing:
It updates the routing information for each machine in the DHT by calling the **makeFT** function on each machine.

(3) Add machine:
It adds a new machine to the DHT, obtaining a unique identifier based on the machine name and inserting it into the ring while maintaining the order

(4) display Ring:
It displays the circular linked list, printing information about each machine in the DHT.

(5) Insert Mac:
It inserts a new machine with the specified identifier and machine name into the DHT, adjusting the circular linked list and redistributing files from the successor machine if necessary.

(6)Delete Machine:
It deletes a machine with the specified identifier from the DHT, redistributing files to the successor machine before removing the target machine from the circular linked list.

(6) Search Machine:
It searches for a machine with a specific identifier in the DHT, returning true if found and false otherwise.

(7) File Insertion:
It inserts a file into the DHT, generating a key based on the file's content and initiating the file insertion process starting from the head machine.

(8) File Deletion:
It deletes a file from the DHT using the specified key, initiating the file deletion process starting from the head machine.

(9) Print B-tree:
Prints the B-tree associated with a specific machine identifier in the DHT.

(10) print Routing Table:

Prints the routing table (Finger Table) associated with a specific machine identifier in the DHT.

3- Info Struct

This file contains:

Class: Bnode represents a node in a B-tree structure. It contains functions which are described below:

(1) Constructor:
It initializes a B-tree node with a specified order and leaf status. It also allocates memory for arrays to store keys and child pointers. It initializes the arrays with NULL values.

(2) Insert to Node:
It inserts a new key into the node at the appropriate position, maintaining sorted order.

(3)Remove From Node:
It removes a key from the node at a given index, readjusting the keys and child pointers accordingly. Returns the removed key.

(3) split Child:
It splits a child node (y) at a specified index, creating a new node (z) and redistributing keys and child pointers. It also handles a special case where the new entry lies directly between the split and split+1.

(4)Insert Non Full:
Inserts a new key into a non-full node (x). If the node is a leaf, inserts the key directly. If the node is not a leaf, continues traversal and splits child nodes where necessary.

Class: Info Each Info instance has a key and a corresponding value. The class includes a constructor allowing initialization with default or specified values. Additionally, a copy constructor is provided for creating a new Info instance based on an existing one, preserving the key and value attributes.

Class: B Trees contains following functions:

(1) Constructor:
Initializes a B-tree with a specified order, setting the root to NULL.
(2)Display In Order:
Displays the B-tree in-order, printing keys and their levels.
(3)duplication:
Checks if a string contains a newline character, indicating duplicates.
(4)exist:

Checks if a key exists in the B-tree.
(5)search:
Searches for a key in the B-tree, returning associated information if found.
(6) insert:
Inserts a new key into the B-tree, handling duplication and node splitting.
(7) delete Key:
Deletes a key from the B-tree, handling duplication, internal node deletion, and adjustments.

(8) Delete internal node:
Deletes a key from an internal node during deletion, handling merging and sibling adjustments.

(9) delete predecessor:
Deletes the predecessor key during deletion, handling cases of child node adjustments.

(10) delete successor:
Deletes the successor key during deletion, handling cases of child node adjustments.

(11) delete merge:
Merges a node with its sibling during deletion, updating keys and children.

(12) delete sibling:
Adjusts nodes by borrowing a key from a sibling during deletion.

(13) Delete Tree:
Recursively deletes the B-tree, freeing memory occupied by keys and nodes.

(13) **delete Entire Tree**:
Deletes the entire B-tree, calls delete Tree function and setting the root to NULL.

(14) **smallest**: Finds and returns the smallest key in the B-tree by traversing leftmost children.

4- **Hashing**

This file provides several utility functions for manipulating hexadecimal strings and performing operations on them.
It contains following functions:

(1) Read File:
Reads the content of a file specified by the filePath parameter. Returns the content as a string or an empty string if the file cannot be opened.

(2)Calculate Unique Chars:
Calculates the minimum number of characters needed for uniqueness based on the total number of nodes in a data structure. The calculation is based on the logarithmic function.

(2) Hash:
Computes the SHA-1 hash of the input string. Returns the hexadecimal representation of the hash.

(3) Get Last Substring:
Extracts the last substring of a specified length from a SHA-1 hash.

(4) pad String:
Pads a string with leading zeros to achieve the desired length.

(5) subtract Hexadecimal Strings:
Subtracts one hexadecimal string (hex2) from another (hex1). Returns the result as a hexadecimal string.

(6) Add Hexadecimal Strings:
Adds two hexadecimal strings (hex1 and hex2). Returns the result as a hexadecimal string.

(7) Power Of Two To Hex:

Computes 2^exponent and returns the result as a hexadecimal string.
Utilizes the **addHexadecimalStrings** function for exponentiation.