Lab Exercise – 16

AIM :: Write a C program to implement various `File Organization Techniques`

Theory :: **File Organization in Operating Systems**

File organization refers to the method or structure used to store, organize, and retrieve records in a file on secondary storage (e.g., hard drives). The choice of file organization technique can significantly affect the efficiency of data retrieval and insertion.

There are various file organization techniques, each suited for specific types of operations and applications.

Key Types of File Organization:

1. Sequential File Organization

- **Definition**: In this method, records are stored one after the other in a sequential manner.
- **Features**:
  - Records are stored in the order they are created or based on some key value.
  - To retrieve a record, the system may need to scan through many records before finding the desired one.
  - Efficient for sequential access but slow for random access.
- **Applications**:
  - Used when records are accessed in a fixed sequence.
  - Suitable for batch processing systems (e.g., payroll, inventory management).
- **Advantages**:
  - Simple to implement.
  - Low overhead for maintaining records.
- **Disadvantages**:
  - Searching can be slow.
  - Modifications require shifting records, which is time-consuming.

2. Direct (Hashed) File Organization

- **Definition**: This method uses a hashing algorithm to compute the address or location of the record in the file, based on the record's key.
- **Features**:
  - Records are stored in random locations based on a hash function.
  - Ideal for quick and direct access to records (constant time lookup).
  - Collisions (when two keys map to the same location) are handled using techniques like linear probing or chaining.
- **Applications**:

    - Used when fast access to individual records is critical (e.g., in databases and index-based systems).
- **Advantages**:
    - Extremely fast access for both reading and writing when collisions are minimal.
- **Disadvantages**:
    - Collisions can lead to slower performance.
    - Requires a good hash function to minimize collisions.
    - Difficult to expand the file size dynamically without significant overhead.

3. Indexed File Organization

- **Definition**: In indexed organization, a separate index is created, which contains key-field pointers to the actual location of the records in the data file.
- **Features**:
    - The index file stores keys and the addresses of corresponding records.
    - For each search, the system first searches the index file, finds the record's location, and then retrieves the record from the data file.
    - Supports both sequential and random access.
- **Applications**:
    - Widely used in database systems, especially when quick access to records based on certain key fields is required.
    - Efficient for scenarios with both high read and write operations.
- **Advantages**:
    - Quick search based on the index.
    - Records can be retrieved without scanning the entire file.
- **Disadvantages**:
    - Index maintenance adds overhead (especially for large datasets).
    - Requires extra storage for the index file.

4. Clustered File Organization

- **Definition**: In this organization method, related records are stored together based on some clustering criteria. These related records are stored on the same block, which reduces disk I/O.
- **Features**:
    - Improves access time by storing related records close to each other.
    - Clustering can be based on one or more fields that define the relationship between records.
- **Applications**:
    - Useful in scenarios where related data is frequently accessed together (e.g., database joins, transaction systems).
- **Advantages**:
    - Speeds up data retrieval for related data sets.
    - Reduces the number of disk I/O operations.
- **Disadvantages**:
    - Complex to manage as the clustering criteria must be carefully defined.
    - Can cause performance degradation if unrelated records are stored together.

5. Multilevel Indexing

- **Definition**: This is an extension of indexed file organization, where a primary index points to several secondary indexes that in turn point to actual records.
- **Features**:
  - Helps organize large databases where a single index file would be too large to efficiently manage.
  - Breaks down the indexing into multiple levels for faster lookups.
- **Applications**:
  - Used in very large-scale database systems (e.g., distributed file systems).
- **Advantages**:
  - Improves search efficiency by using a hierarchical structure of indexes.
  - Suitable for large datasets.
- **Disadvantages**:
  - Increased complexity in managing multiple levels of indexes.
  - Higher storage overhead due to multiple index files.

Factors to Consider When Choosing a File Organization Technique:

- **Access Type**:
  - Sequential access vs. random access. Sequential organization is better for batch processing, while direct and indexed methods are preferable for random access.
- **Frequency of Operations**:
  - How often records will be inserted, deleted, updated, or retrieved. Direct and indexed file organizations work better for high-frequency access scenarios.
- **File Size**:
  - Larger files can benefit from indexed or multi-level indexed file organizations to avoid performance degradation.
- **Collision Handling (in Direct Organization)**:
  - For hashed files, an effective collision resolution strategy is essential.
- **Storage Overhead**:
  - Indexes and hash tables require additional storage space. Choose accordingly based on available resources.

Source_Code ::

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX 100
#define HASH_SIZE 10
```

```
// Structure for a record in a file
struct Record
{
  int id;
  char name[20];
};

// Hashing function for Direct File Organization
int hashFunction(int id)
{
  return id % HASH_SIZE;
}

// Sequential File Organization: Insert record
void sequentialInsert(FILE *file, struct Record rec)
{
  fwrite(&rec, sizeof(struct Record), 1, file);
}

// Sequential File Organization: Display records
void sequentialDisplay(FILE *file)
{
  struct Record rec;
  rewind(file); // Move pointer to the beginning of the file
  while (fread(&rec, sizeof(struct Record), 1, file))
  {
    printf("ID: %d, Name: %s\n", rec.id, rec.name);
  }
}

// Direct File Organization: Insert record (using hashing)
void directInsert(struct Record hashTable[], struct Record rec)
{
  int index = hashFunction(rec.id);
  while (hashTable[index].id != -1)
  {
    index = (index + 1) % HASH_SIZE;
  }
  hashTable[index] = rec;
}

// Direct File Organization: Display records
```

```
void directDisplay(struct Record hashTable[])
{
  for (int i = 0; i < HASH_SIZE; i++)
  {
    if (hashTable[i].id != -1)
    {
      printf("ID: %d, Name: %s (Index: %d)\n", hashTable[i].id, hashTable[i].name, i);
    }
  }
}

// Indexed File Organization: Insert record
void indexedInsert(FILE *dataFile, FILE *indexFile, struct Record rec)
{
  fseek(dataFile, 0, SEEK_END);    // Move to end of data file
  long position = ftell(dataFile); // Get current position in file
  fwrite(&rec, sizeof(struct Record), 1, dataFile);

  // Write index entry
  fwrite(&rec.id, sizeof(int), 1, indexFile);
  fwrite(&position, sizeof(long), 1, indexFile);
}

// Indexed File Organization: Display records
void indexedDisplay(FILE *dataFile, FILE *indexFile)
{
  int id;
  long position;
  struct Record rec;

  rewind(indexFile); // Move to the beginning of the index file
  while (fread(&id, sizeof(int), 1, indexFile) && fread(&position, sizeof(long), 1, indexFile))
  {
    fseek(dataFile, position, SEEK_SET); // Move to the position in data file
    fread(&rec, sizeof(struct Record), 1, dataFile);
    printf("ID: %d, Name: %s\n", rec.id, rec.name);
  }
}

int main()
{
  printf("\n5C6 - Amit Singhal (11614802722)\n");
```

```
FILE *seqFile, *dataFile, *indexFile;
struct Record hashTable[HASH_SIZE];
struct Record rec;
int choice, id, index;
char name[20];

// Initialize hash table for direct file organization
for (int i = 0; i < HASH_SIZE; i++)
{
  hashTable[i].id = -1; // Empty slot
}

// Open files for sequential and indexed file organization
seqFile = fopen("sequential.dat", "wb+");
dataFile = fopen("data.dat", "wb+");
indexFile = fopen("index.dat", "wb+");

if (!seqFile || !dataFile || !indexFile)
{
  printf("Error opening file!\n");
  return 1;
}

do
{
  printf("\nFile Organization Menu:\n");
  printf("1. Sequential Insert\n");
  printf("2. Sequential Display\n");
  printf("3. Direct Insert (Hashing)\n");
  printf("4. Direct Display (Hashing)\n");
  printf("5. Indexed Insert\n");
  printf("6. Indexed Display\n");
  printf("7. Exit\n");
  printf("Enter your choice: ");
  scanf("%d", &choice);

  switch (choice)
  {
  case 1:
    printf("Enter ID: ");
    scanf("%d", &rec.id);
    printf("Enter Name: ");
    scanf("%s", rec.name);
```

```
      sequentialInsert(seqFile, rec);
      break;

    case 2:
      sequentialDisplay(seqFile);
      break;

    case 3:
      printf("Enter ID: ");
      scanf("%d", &rec.id);
      printf("Enter Name: ");
      scanf("%s", rec.name);
      directInsert(hashTable, rec);
      break;

    case 4:
      directDisplay(hashTable);
      break;

    case 5:
      printf("Enter ID: ");
      scanf("%d", &rec.id);
      printf("Enter Name: ");
      scanf("%s", rec.name);
      indexedInsert(dataFile, indexFile, rec);
      break;

    case 6:
      indexedDisplay(dataFile, indexFile);
      break;

    case 7:
      printf("Exiting...\n");
      break;

    default:
      printf("Invalid choice!\n");
      break;
    }
  } while (choice != 7);

  fclose(seqFile);
  fclose(dataFile);
```

```
    fclose(indexFile);

    return 0;
}
```

Output ::

```
singhal-amit@singhal-amit-ThinkPad-T430:~$ cd Desktop/
singhal-amit@singhal-amit-ThinkPad-T430:~/Desktop$ vi amit.c
singhal-amit@singhal-amit-ThinkPad-T430:~/Desktop$ gcc amit.c -o a
singhal-amit@singhal-amit-ThinkPad-T430:~/Desktop$ ./a

5C6 - Amit Singhal (11614802722)

File Organization Menu:
1. Sequential Insert
2. Sequential Display
3. Direct Insert (Hashing)
4. Direct Display (Hashing)
5. Indexed Insert
6. Indexed Display
7. Exit
Enter your choice: 1
Enter ID: 116
Enter Name: Amit

File Organization Menu:
1. Sequential Insert
2. Sequential Display
3. Direct Insert (Hashing)
4. Direct Display (Hashing)
5. Indexed Insert
6. Indexed Display
7. Exit
Enter your choice: 1
Enter ID: 122
Enter Name: Yash

File Organization Menu:
1. Sequential Insert
2. Sequential Display
3. Direct Insert (Hashing)
4. Direct Display (Hashing)
5. Indexed Insert
6. Indexed Display
7. Exit
Enter your choice: 2
ID: 116, Name: Amit
ID: 122, Name: Yash
```

```
File Organization Menu:
1. Sequential Insert
2. Sequential Display
3. Direct Insert (Hashing)
4. Direct Display (Hashing)
5. Indexed Insert
6. Indexed Display
7. Exit
Enter your choice: 3
Enter ID: 11
Enter Name: Divyam

File Organization Menu:
1. Sequential Insert
2. Sequential Display
3. Direct Insert (Hashing)
4. Direct Display (Hashing)
5. Indexed Insert
6. Indexed Display
7. Exit
Enter your choice: 4
ID: 11, Name: Divyam (Index: 1)

File Organization Menu:
1. Sequential Insert
2. Sequential Display
3. Direct Insert (Hashing)
4. Direct Display (Hashing)
5. Indexed Insert
6. Indexed Display
7. Exit
Enter your choice: 5
Enter ID: 105
Enter Name: Shaswat

File Organization Menu:
1. Sequential Insert
2. Sequential Display
3. Direct Insert (Hashing)
4. Direct Display (Hashing)
5. Indexed Insert
6. Indexed Display
7. Exit
Enter your choice: 5
Enter ID: 666
Enter Name: Nitin
```

```
File Organization Menu:
1. Sequential Insert
2. Sequential Display
3. Direct Insert (Hashing)
4. Direct Display (Hashing)
5. Indexed Insert
6. Indexed Display
7. Exit
Enter your choice: 6
ID: 105, Name: Shaswat
ID: 666, Name: Nitin

File Organization Menu:
1. Sequential Insert
2. Sequential Display
3. Direct Insert (Hashing)
4. Direct Display (Hashing)
5. Indexed Insert
6. Indexed Display
7. Exit
Enter your choice: 7
Exiting...
singhal-amit@singhal-amit-ThinkPad-T430:~/Desktop$
```

File Structure ::