

Ex. No: 19

Date:

INDEXED FILE ALLOCATION

Problem Statement:

To write a python program to implement Indexed file allocation method

Problem Description:

Instead of maintaining a file allocation table of all the disk pointers, Indexed allocation scheme stores all the disk pointers in one of the blocks called as indexed block. Indexed block doesn't hold the file data, but it holds the pointers to all the disk blocks allocated to that particular file. Directory entry will only contain the index block address.

Algorithm:

Step 1: Initialize the disk and data structures

Read the number of blocks in the disk from the user and store it in 'num_block'.

- Create a list 'disk' of length 'num_block' and initialize it with -1 to represent free blocks.
- Create an array 'arr' to store file block mappings.
- Create a list 'file_map' to store file names.
- Create a list 'map_idx' to store the index block for each file.
- Create a list 'sz' to store the total number of blocks in each file.
- Initialize 'tot_file' to 0 to keep track of the total number of files.

Step 2: Start an infinite loop to display the menu and handle user choices.

Step 3: Print the menu options: To add a file, to print the directory, to exit

Step 4: Handle user choice

- Read the user's choice (integer) and store it in the 'choice' variable.
- Use conditional statements to handle different choices:

If 'choice' is 1:

- Read the file name, indexed block, and the total number of blocks in the file from the user.
- Validate the indexed block to ensure it is not occupied by another file.
- Find and allocate free blocks for the file's data blocks.
- Update data structures to store the file's information (name, index block, and size).
- Increment 'tot_file' to reflect the addition of a new file.

If 'choice' is 2:

- Print the directory showing file names, index blocks, and the blocks stored for each file.

If 'choice' is 3:

- Exit the program.

If 'choice' is not in the range 1-3, print "Invalid Input."

Step 5: Exit the program

Code:

```
def print_menu():
    print("Enter:")
    print("1. To add File")
    print("2. To Print Directory")
    print("3. To exit")
if __name__ == "__main__":
    num_block = int(input("Enter the number of blocks in the disk: "))
    disk = [-1] * num_block
    arr = [[None] * 100000 for _ in range(1000)]
    file_map = [None] * 100000
    map_idx = [0] * 100000
    sz = [0] * 100000
    tot_file = 0
    print("Welcome to the indexed File")
    while True:
        print_menu()
        choice = int(input())
        if choice == 1:
            file_name, i_block, t_block = input("Enter File name, indexed block, and total
number of blocks in the file: ").split()
            i_block, t_block = int(i_block), int(t_block)
            if i_block < 0 or i_block >= num_block or disk[i_block] != -1:
```

```

        print("Index Block is not empty or invalid")
        continue
    free_block = []
    j = 0
    cnt = 0
    disk[i_block] = tot_file
    for i in range(t_block):
        while j < num_block:
            if disk[j] == -1:
                free_block.append(j)
                cnt += 1
                j += 1
                break
            j += 1
    if cnt == t_block:
        for i in range(t_block):
            arr[tot_file][i] = free_block[i] # Store free block index
            disk[free_block[i]] = tot_file
        print()
        file_map[tot_file] = file_name
        map_idx[tot_file] = i_block
        sz[tot_file] = t_block
        tot_file += 1
    else:
        disk[i_block] = -1
        print("Not Enough free Space")
elif choice == 2:
    print("File name      Index Block   Block Stored")
    for i in range(tot_file):
        print(f"{file_map[i]:<16} {map_idx[i]:<16}", end="")

```

```

        for j in range(sz[i]):
            print(f"{arr[i][j]}", end="")

            if j < sz[i] - 1:
                print(" --> ", end="")

        print()

    elif choice == 3:
        break

    else:
        print("Invalid Input")

```

Output:

```

Enter the number of blocks in the disk: 30
Welcome to the indexed File
Enter:
1. To add File
2. To Print Directory
3. To exit
1
Enter File name, indexed block, and total number of blocks in the file: file1.txt 15 6

Enter:
1. To add File
2. To Print Directory
3. To exit
2
File name      Index Block    Block Stored
file1.txt      15             0 --> 1 --> 2 --> 3 --> 4 --> 5
Enter:
1. To add File
2. To Print Directory
3. To exit
1
Enter File name, indexed block, and total number of blocks in the file: file2 29 23
Not Enough free Space
Enter:
1. To add File
2. To Print Directory
3. To exit
1
Enter File name, indexed block, and total number of blocks in the file: file2.txt 29 22

```

```
Enter:
1. To add File
2. To Print Directory
3. To exit
2
File name      Index Block   Block Stored
file1.txt      15             0 --> 1 --> 2 --> 3 --> 4 --> 5
file2.txt      29             6 --> 7 --> 8 --> 9 --> 10 --> 11 --> 12 --> 13 --> 14 --> 16 --> 17 --> 18 --> 19 --> 20 --> 21 --> 22 --> 23 --> 24 --> 25 --> 26 --> 27
--> 28
Enter:
1. To add File
2. To Print Directory
3. To exit
1
Enter File name, indexed block, and total number of blocks in the file: file3.txt 22 5
Index Block is not empty or invalid
Enter:
1. To add File
2. To Print Directory
3. To exit
3
```

Ex. No: 19

Date:

FILE ALLOCATION TECHNIQUES USING LINKED LIST

Problem Statement:

To implement file allocation using linked list.

Problem Description:

Linked File Allocation is a Non-contiguous memory allocation method where the file is stored in random memory blocks and each block contains the pointer (or address) of the next memory block as in a linked list. The starting memory block of each file is maintained in a directory and the rest of the file can be traced from that starting block.

Algorithm:

1. Initialize data structures to represent disk blocks, file names, and file metadata.
2. Display a menu for the user with options to add a file, print the directory, or exit.
3. In an infinite loop, wait for the user to choose an option.
4. If the user chooses to add a file:
 - Prompt for the file name, starting block, and total blocks.
 - Check if the file name exists, the starting block is valid and available, and if there are enough free blocks.
 - Allocate blocks to the file, update data structures, and keep track of the total files.
5. If the user chooses to print the directory:
 - Display the list of file names and their allocated blocks.
6. If the user chooses to exit, terminate the program.
7. If the user provides an invalid option, display an error message.
8. End the loop when the user exits the program.

Code:

```
import random
```

```
class Node:
```

```
    def __init__(self, val): self.ptr  
        = None
```

```
self.val = val
```

```
arr = [None] * 100 fileMap =
```

```
[None] * 100 mapIdx = [0] *
```

```
100
```

```
sz = [0] * 100
```

```
def printMenu(): print("Enter:")
```

```
    print("1. To add File")
```

```
    print("2. To Print Directory") print("3. To
```

```
    exit")
```

```
def main():
```

```
    numBlock = int(input("Enter the number of Block in disk: ")) disk = [-1] *
```

```
    numBlock
```

```
    totFile = 0 file=[]
```

```
    blocks=[] count=0
```

```
    while True:
```

```
        printMenu()
```

```
        choice = int(input())
```

```
        if choice == 1:
```

```
            fName = input("Enter File name: ") if
```

```
            fName in file:
```

```
                print("File already exists") continue
```

```
            sBlock = int(input("Enter the starting block: "))
```

```
            if sBlock < 0 or sBlock >= numBlock or disk[sBlock] != -1:
```

```

        print("Start Block is not empty or invalid") continue
tBlock = int(input("Enter the total number of blocks in the
file: "))
if count>numBlock:
    print("Enter valid no of blocks!") continue
file.append(fName) freeBlock =
[sBlock] blocks.append(sBlock)

for _ in range(tBlock - 1): while
    True:
        randomBlock = random.randint(0, numBlock - 1)
        if disk[randomBlock] == -1 and randomBlock not in disk and
randomBlock not in blocks:
            freeBlock.append(randomBlock)
            disk[randomBlock] = totFile
            count+=1
            blocks.append(randomBlock) break

head = None
for block in freeBlock: idx =
    Node(block) idx.ptr =
    None
    if head is None: arr[totFile] =
        idx head = idx
    else:
        head.ptr = idx head =
        idx

```



```
fileMap[totFile] = fName
sz[totFile] = tBlock totFile += 1
if count==numBlock: break
```

```
elif choice == 2:
```

```
    print("File name          Block Stored")
    for i in range(totFile):
        print(f" {fileMap[i]}          ",end="")
        head = arr[i]
        while head is not None: print(f" {head.val} -
            >",end="") head = head.ptr
    print("NULL")
```

```
elif choice == 3: break
```

```
else:
```

```
    print("Invalid Input")
```

```
if __name__ == "__main__":
    main()
```

Output:

```
Enter the number of Block in disk: 30
Enter:
1. To add File
2. To Print Directory
3. To exit
1
Enter File name: file1.txt
Enter the starting block: 4
Enter the total number of blocks in the file: 7
Enter:
1. To add File
2. To Print Directory
3. To exit
2
File name      Block Stored
file1.txt      4 ->10 ->13 ->5 ->9 ->14 ->18 ->NULL
Enter:
1. To add File
2. To Print Directory
3. To exit
1
Enter File name: file2.txt
Enter the starting block: 8
Enter the total number of blocks in the file: 7
Enter:
1. To add File
2. To Print Directory
3. To exit
2
File name      Block Stored
file1.txt      4 ->10 ->13 ->5 ->9 ->14 ->18 ->NULL
file2.txt      8 ->25 ->2 ->7 ->20 ->28 ->6 ->NULL
Enter:
1. To add File
2. To Print Directory
3. To exit
1
Enter File name: file2.txt
File already exists
```

File already exists

Enter:

1. To add File
2. To Print Directory
3. To exit

1

Enter File name: file3.txt

Enter the starting block: 0

Enter the total number of blocks in the file: 6

Enter:

1. To add File
2. To Print Directory
3. To exit

2

File name Block Stored

file1.txt 4 ->10 ->13 ->5 ->9 ->14 ->18 ->NULL

file2.txt 8 ->25 ->2 ->7 ->20 ->28 ->6 ->NULL

file3.txt 0 ->26 ->15 ->3 ->24 ->19 ->NULL

Enter:

1. To add File
2. To Print Directory
3. To exit

1

Enter File name: file4.txt

Enter the starting block: 20

Start Block is not empty or invalid

Enter:

1. To add File
2. To Print Directory
3. To exit

1

Enter File name: file4.txt

Enter the starting block: 1

Enter the total number of blocks in the file: 5

Enter:

1. To add File
2. To Print Directory
3. To exit

2

File name Block Stored

file1.txt 4 ->10 ->13 ->5 ->9 ->14 ->18 ->NULL

file2.txt 8 ->25 ->2 ->7 ->20 ->28 ->6 ->NULL

file3.txt 0 ->26 ->15 ->3 ->24 ->19 ->NULL

file4.txt 1 ->12 ->22 ->21 ->17 ->NULL

Ex. No: 19

Date:

SEQUENTIAL/CONTIGUOUS FILE ALLOCATION METHOD

Problem Statement:

To write a python program to implement Sequential/Contiguous File Allocation Techniques.

Problem Description:

In this allocation strategy, each file occupies a set of contiguous blocks on the disk. This strategy is best suited. For sequential files, the file allocation table consists of a single entry for each file. It shows the filenames, starting block of the file and size of the file. The main problem with this strategy is, it is difficult to find the contiguous free blocks in the disk and some free blocks could happen between two files.

Algorithm:

1.Initialize a list called disk with num_blocks elements, all initialized to -1, representing the storage blocks on the virtual disk.

2.Create dictionaries (file_map, map_idx, map_end_idx) to store file information, including file names, starting blocks, and ending blocks.

Initialize tot_file to 0, representing the total number of allocated files.

3.Start an infinite loop for user interaction.

 Inside the loop, display a menu of options for the user to choose from.

 (1: Add File, 2: Print Directory, 3: Exit)

 Prompt the user to enter their choice by accepting an integer input.

 Based on the user's choice, execute the corresponding functionality.

4.If the user chooses to add a file:

 Enter the File name Starting block Total number of blocks required for the file.

5.Initialize a flag (flag) to 0.

6.Check if the requested blocks are available and if the file size can fit within the available space

7.Iterate through the disk from the start_block to check block availability. If a block is already allocated, break out of the loop.

8.If the loop completes without breaking and total_blocks are greater than 0, set flag to 1.

9.If flag is 1 (allocation is possible):

Allocate the blocks on the disk with a unique file identifier (an integer) for the file.
Store the file information in the dictionaries (file_map, map_idx, map_end_idx) and update tot_file.

10. Print a message indicating the allocated blocks and their range.

11. If flag is 0 (allocation is not possible): Display an error message.

12. Print Directory : Display the directory information in a well-formatted table. Include file names, starting blocks, ending blocks, and the blocks occupied by each file (presented as a range).

13. Exit Program : If the user chooses to exit the program, break out of the loop to terminate the program.

Code:

```
def print_menu():  
    print("\nEnter:")  
    print("1. To add File")  
    print("2. To Print Directory")  
    print("3. To exit")  
  
if __name__ == "__main__":  
    num_blocks = int(input("Enter the number of Blocks in the disk: "))  
    disk = [-1] * num_blocks  
    file_map = {}  
    map_idx = {}  
    map_end_idx = {}  
    tot_file = 0  
  
    print("Welcome to the Sequential File")  
  
    while True:  
        print_menu()  
        choice = int(input("Enter your choice: "))
```

```

if choice == 1:
    file_name = input("Enter File name: ")
    start_block, total_blocks = map(int, input("Enter starting block and total number of
blocks in the file: ").split())
    flag = 0

    for i in range(start_block, num_blocks):
        if disk[i] != -1:
            break
        if start_block + total_blocks - 1 == i:
            flag = 1
            break

    if total_blocks == 0:
        flag = 0

    if flag == 1:
        occupied_blocks = [str(block) for block in range(start_block, start_block +
total_blocks)]
        disk[start_block:start_block + total_blocks] = [tot_file] * total_blocks

        file_map[tot_file] = file_name
        map_idx[tot_file] = start_block
        map_end_idx[tot_file] = start_block + total_blocks - 1
        tot_file += 1

        print(f"File '{file_name}' allocated from block {start_block} to {start_block +
total_blocks - 1}.")
    else:
        print("Something Went Wrong, either someone has occupied that space or out of
bounds.")

    elif choice == 2:
        print("\n{:<15} {:<15} {:<15} {:<15}".format("File name", "Start Block", "End
Block", "Blocks Occupied"))

```

```

        for i in range(tot_file):

            start_block = map_idx[i]

            end_block = map_end_idx[i]

            occupied_blocks = [str(block) for block in range(start_block, end_block + 1)]

            print("{:<15} {:<15} {:<15} {:<15}".format(file_map[i], start_block, end_block, ' -
-> '.join(occupied_blocks)))

        elif choice == 3:

            break

    else:

        print("Invalid Input")

```

Output:

```

Enter the number of Blocks in the disk: 30
Welcome to the Sequential File

Enter:
1. To add File
2. To Print Directory
3. To exit
Enter your choice: 1
Enter File name: file1
Enter starting block and total number of blocks in the file: 18 9
File 'file1' allocated from block 18 to 26.

Enter:
1. To add File
2. To Print Directory
3. To exit
Enter your choice: 2

File name      Start Block    End Block      Blocks Occupied
file1          18              26            18 --> 19 --> 20 --> 21 --> 22 --> 23 --> 24 --> 25 --> 26

Enter:
1. To add File
2. To Print Directory
3. To exit
Enter your choice: 1
Enter File name: file2
Enter starting block and total number of blocks in the file: 27 3
File 'file2' allocated from block 27 to 29.

Enter:
1. To add File
2. To Print Directory
3. To exit
Enter your choice: 2

File name      Start Block    End Block      Blocks Occupied
file1          18              26            18 --> 19 --> 20 --> 21 --> 22 --> 23 --> 24 --> 25 --> 26
file2          27              29            27 --> 28 --> 29

Enter:
1. To add File
2. To Print Directory
3. To exit
Enter your choice: 1

```

```

Enter File name: file3
Enter starting block and total number of blocks in the file: 12 6
File 'file3' allocated from block 12 to 17.

Enter:
1. To add File
2. To Print Directory
3. To exit
Enter your choice: 2

File name      Start Block    End Block      Blocks Occupied
file1          18             26             18 --> 19 --> 20 --> 21 --> 22 --> 23 --> 24 --> 25 --> 26
file2          27             29             27 --> 28 --> 29
file3          12             17             12 --> 13 --> 14 --> 15 --> 16 --> 17

Enter:
1. To add File
2. To Print Directory
3. To exit
Enter your choice: 1
Enter File name: file4
Enter starting block and total number of blocks in the file: 2 10
File 'file4' allocated from block 2 to 11.

Enter:
1. To add File
2. To Print Directory
3. To exit
Enter your choice: 2

File name      Start Block    End Block      Blocks Occupied
file1          18             26             18 --> 19 --> 20 --> 21 --> 22 --> 23 --> 24 --> 25 --> 26
file2          27             29             27 --> 28 --> 29
file3          12             17             12 --> 13 --> 14 --> 15 --> 16 --> 17
file4          2              11             2 --> 3 --> 4 --> 5 --> 6 --> 7 --> 8 --> 9 --> 10 --> 11

Enter:
1. To add File
2. To Print Directory
3. To exit
Enter your choice: 1
Enter File name: file5
Enter starting block and total number of blocks in the file: 0 6
Something Went Wrong, either someone has occupied that space or out of bounds.

```