



VIT[®]
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

PROJECT REPORT

Comparative Study of File Allocation Method

GROUP MEMBERS:

- 1) 17BCI0133-Sujeeth ES
- 2) 17BCE0529- Gowtham.P

FACULTY:- Mr. Saleem Durai

SLOT:- F1

INDEX :

- ABSTRACT
- INTRODUCTION
- FILE ALLOCATION METHODS
- SOURCE CODE
- SNAPSHOTS
- CONCLUSION
- REFERENCES

ABSTRACT

Any existing data processing system may have a file allocation table which is used to store files in an organized and systematic way and is supported by all virtually existing operating systems. It generates good performance and has high strength and therefore is being adopted these days by most personal computers as well as home computers. In the following paper, an introduction to file allocation table is given along with a brief about the different types of memory allocations it has and a comparative study of all the file allocation methods and among those which one is optimal. Along with that the source code for creating, printing blocks table and contents of files, deleting and also printing updated files after deletion is given and the required outputs are shown in form of screenshots.

INTRODUCTION

File Allocation Table[1] is also abbreviated as FAT. It is a table which is maintained by the operating system on the hard disc and it has and is used to enter 12 or 16 bits entry per cluster. It is named after its method of organisation. 2 separate copies of files are kept in case one of them is damaged. A volume formatted with the FAT[1] is allocated into clusters and the cluster should be in powers of 2. It can commonly be found in the flash memory, digital camera and other portable devices. It was originally designed to lessen the amount of time taken to seek on the operating system and also reduce the wear and tear on the hard disc.

FAT16 was introduced in 1983 by IBM and then later on in 1997 Microsoft introduced FAT32. A FAT[1] file has basically 4 different sections-the boot sector which comprises the boot loader code or bootstrap program in order to start a PC system, then the FAT region which has 2 copies of the FAT for redundancy checking, then the Data region where the directory data and files are stored. The last part is the Root Directory region which is a directory table that contains the necessary data about the directories and files. In this way FAT[1] is used to keep a record of all files as well as point towards the cluster containing the files.

FILE ALLOCATION METHODS

File Allocation[2] method is used to effectively utilize disk space available by accommodating space for files. Operating system allocates the disk space for the files by using different approaches. It's divided into three broad categories: Contiguous Allocation[3], Indexed Allocation[4] and Linked allocation[5]. The main objective to adapt these 3 approaches is to make sure there efficient utilization of disk space and speedy access to the files. These three methods use distinct methods to effectively use disk space which has its own pros and cons.

Comparative Study Of Three Approaches

1. Contiguous Allocation Method –

By this approach, continuous blocks of disk spaces is assigned to each and every file. It uses the concept of first fit or best fit. It's the simplest allocation method.

Location of file is identified by the starting address of the block followed by the length of file. These to details are crucial to work with file.

File	Starting Address	Length
A	2	3
B	6	2

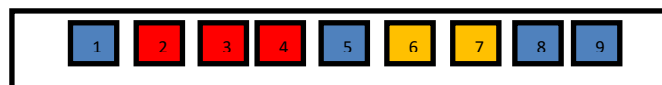


Fig 1:contiguous Allocation Method

Advantages –For contiguous memory allocation we can easily access the file Sequential Access and Direct Access. Multiple reading is not required. Can be read by a single operation since it's allocated in continuous blocks resulting in excellent operation. Number of disk seeks for accessing the file is minimal so it's extremely fast.

Disadvantages-The major con is that the maximum size of the file has to be limited at the starting of creation itself. Most of the disk space is wasted due to external fragmentation leading to inefficient use of disk space. Compaction can be applied as a solution to external fragmentation. It's a very expensive solution to the problem.

2. Linked Allocation Method-For File allocation method, each file is not allocated in continuous memory blocks. The disk blocks are placed anywhere on the disk. Each file is a linked list of blocks. In this approach, the directory contains the starting address and ending address of the file which are pointers. Then each block contains a pointer that points to the next block.

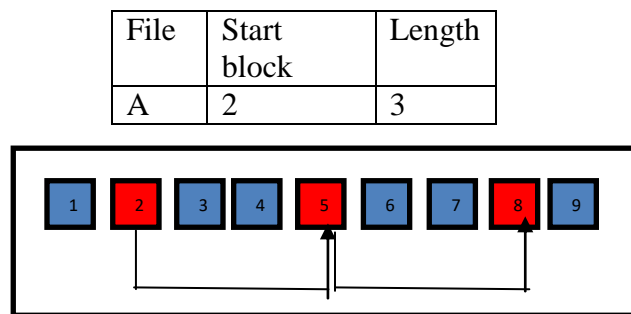


Fig 2: Linked Allocation Method

Advantages – The waste of memory is minimized and maximum disk space utilization is implemented. External fragmentation is not present in this method of allocation. Only internal fragmentation is possible in the last block. It's not necessary to specify the file size during creation.

Disadvantages-Space is occupied for storing pointers leading to extra overhead. Access time is much greater for linked list allocation since the files are located in random order. This method doesn't support direct access only supports sequential access. For finding a block of file it has to implement sequential access from beginning of the file and follow the pointers until the block of the file is identified.

3.Indexed Allocation Method-All shortcomings of linked allocation method and contiguous allocation method is overcome by indexed allocation method. In this approach, all the pointers are brought together into one location known as the indexed block. All the pointers in index block is set to null.

Advantages-This method supports direct access resulting in faster access of the files also supports sequential method of accessing the file. In this method external fragmentation is absent. Directory just keeps track of the starting block address. Free blocks in the disk is used efficiently.

Disadvantages-The index table should be present in the memory. The index block should be large to hold pointers. Searching for entry in index table is tedious procedure.

File	Index block
A	8

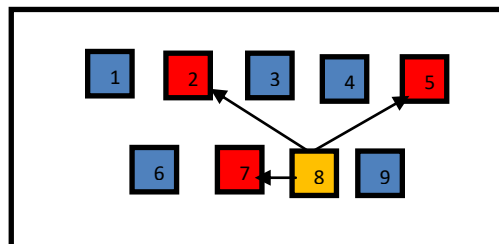


Fig 3:Indexed Allocation Method

III.SOURCE CODE

```

#include <stdio.h>

#include <malloc.h>

#include <stdlib.h>

#include <conio.h>

#include <string.h>

```

```
ints_b[10][2],empty[10]; //Stores size and number of blocks

intmem=500; //keeps track of free memory

char *block[10]; //store address of the blocks

int *address[10]; //stores address of files

char name[10][30]; //stores name of the file

void create() //to create a file

{
    charfname[30],c;
    int size=1,i;
    printf("Enter .txt file name\n");
    scanf("%s",fname);
    FILE *inputf;
    inputf = fopen(fname,"r");
    if (inputf == NULL)
    {
        printf("\nFile unable to open ");
        exit(0);
    }
    rewind(inputf);
    c=fgetc(inputf);
    while(c!=EOF)
    {
        c=fgetc(inputf);
        size=size+1;
    }
```

```
printf("The size of given file is : %d\n", size);
if(mem>=size)
{
int n=1,parts=0,m=1;
while(address[n]!=0)
n++;
strcpy(name[n],fname);
s_b[n][1]=size;
intbnum=size/50;
if(size%50!=0)
bnum=bnum+1;
s_b[n][2]=bnum;
mem=mem-(bnum*50);
int *bfile=(int*)malloc(bnum*(sizeof(int)));
address[n]=bfile;
printf("Number of blocks required: %d\n",bnum);
rewind(inputf);
c = fgetc(inputf);
while(parts!=bnum&& c!=EOF)
{
int k=0;
if(empty[m]==0)
{
char *temp=block[m];
while(k!=50)
```

8

```
{  
    *temp=c;  
    c=fgetc(inputf);  
    temp++;  
    k=k+1;  
}  
*(bfile+parts)=m;  
parts=parts+1;  
empty[m]=1;  
}  
else  
    m=m+1;  
}  
printf("File created\n");  
printf("\n");  
fclose(inputf);  
}  
else  
    printf("Not enough memory\n");  
}  
intfilenum(char fname[30])  
{  
    int i=1,fnum=0;  
    while(name[i])  
    {
```



```
if(strcmp(name[i], fname) == 0)
{
    fnum=i;
    break;
}
i++;
}
returnfnum;
}

void blocks()
{
    int i;
    printf(" Block address empty/free\n");
    for(i=1;i<=10;i++)
        printf("%d. %d - %d\n",i,block[i],empty[i]);
    printf("\n");
}

void file()
{
    int i=1;
    printf("File name size address\n");
    for(i=1;i<=10;i++)
    {
        if(address[i]!=0)
            printf("%s %d %d\n",name[i],s_b[i][1],address[i]);
    }
}
```

10

```
}  
  
printf("\n");  
  
}  
  
void print()  
{  
  
    charfname[30];  
  
    int i=1,j,k,fnum=0;  
  
    printf("Enter the file name: ");  
  
    scanf("%s",fname);  
  
    fnum=filenum(fname);  
  
    if(fnum!=0&& address[fnum]!=0)  
    {  
  
        int *temp;  
  
        temp=address[fnum];  
  
        printf("Content of the file %s is:\n",name[fnum]);  
  
        int b=(s_b[fnum][2]);  
  
        for(j=0;j<b;j++)  
        {  
  
            int s=*(temp+j);  
  
            char *prt=block[s];  
  
            for(k=0;k<50;k++)  
            {  
  
                printf("%c",*prt);  
  
                prt++;  
  
            }  
        }  
    }  
}
```

11

```
    }  
    printf("\n");  
    printf("\n");  
    }  
    else  
    printf("File not available:/n");  
    }  
    void change()  
    {  
    charfname[30];  
    int i=1,j,k,fnum=0;  
    printf("Enter the file name: ");  
    scanf("%s",fname);  
    fnum=filenum(fname);  
    if(fnum==0)  
    printf("File not available:/n");  
    else  
    {  
    int *temp=address[fnum];  
    int b=(s_b[fnum][2]);  
    mem=mem+b*50;  
    for(j=0;j<b;j++)  
    {  
    int s=*(temp+j);  
    empty[s]=0;
```

12

```
}  
address[fnum]=0;  
}  
printf("\n");  
}  
  
int main()  
{  
char*buffer =(char*)malloc(500); //Memory created-500 bytes  
intchoice,i;  
char *temp;  
if (buffer == NULL)  
{  
fputs ("Memory error",stderr);  
exit (2);  
}  
temp=buffer;  
block[1]=buffer;  
empty[1]=0;  
for(i=2;i<=10;i++)  
{  
block[i]=block[i-1]+50;  
empty[i]=0;  
}  
while(1)  
{
```

```
printf("1.Create a new file\n");
printf("2.Delete a file\n");
printf("3.Print a file \n");
printf("4.Display FAT table\n");
printf("5.Display Block Details\n");
printf("6.Exit.\n");
printf("Enter your choice: ");
scanf("%d",&choice);
switch(choice)
{
case 1:
create();
break;
case 2:
change();

break;
case 3:
print();
break;
case 4:
file();
break;
case 5:
blocks();
```

14

```
break;
```

```
case 6:
```

```
exit(1);
```

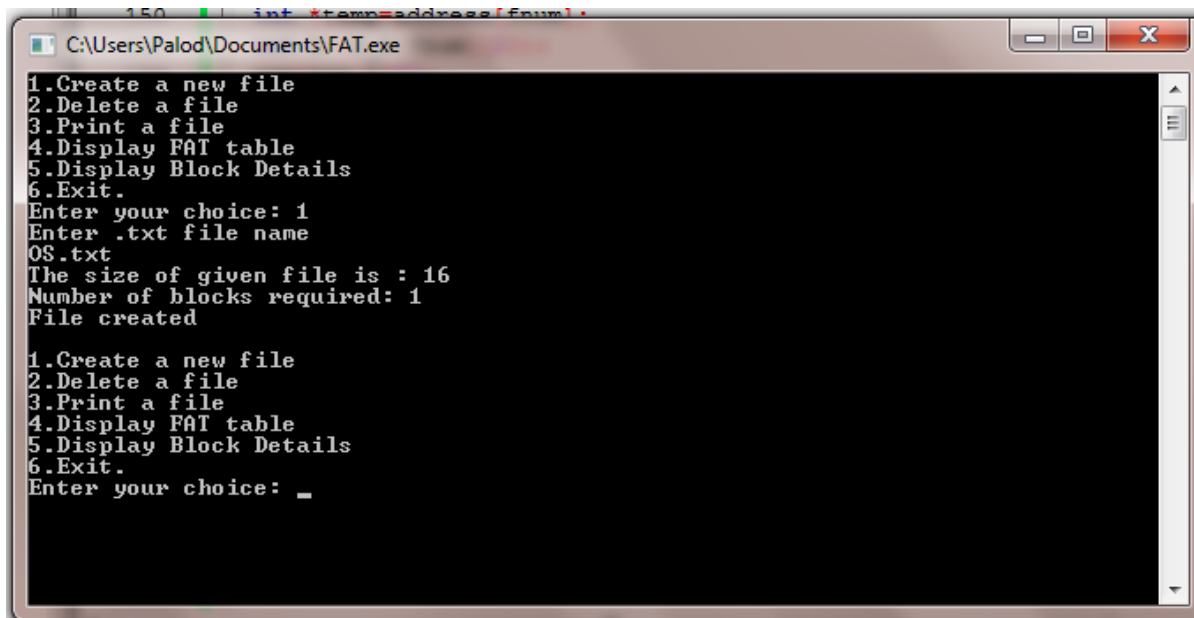
```
}
```

```
}
```

```
return 0;
```

```
}
```

1.TO CREATE FILE



```
150 int *temp=address[fnum];
C:\Users\Palod\Documents\FAT.exe
1.Create a new file
2.Delete a file
3.Print a file
4.Display FAT table
5.Display Block Details
6.Exit.
Enter your choice: 1
Enter .txt file name
OS.txt
The size of given file is : 16
Number of blocks required: 1
File created

1.Create a new file
2.Delete a file
3.Print a file
4.Display FAT table
5.Display Block Details
6.Exit.
Enter your choice: _
```

2.TO PRINT FAT TABLE

```

C:\Users\Palod\Documents\FAT.exe
Enter your choice: 1
Enter .txt file name
OS2.txt
The size of given file is : 9
Number of blocks required: 1
File created

1.Create a new file
2.Delete a file
3.Print a file
4.Display FAT table
5.Display Block Details
6.Exit.
Enter your choice: 4
File name size address
OS.txt 16 11538192
OS2.txt 9 11538800

1.Create a new file
2.Delete a file
3.Print a file
4.Display FAT table
5.Display Block Details
6.Exit.
Enter your choice:

```

3.TO PRINT BLOCK TABLE

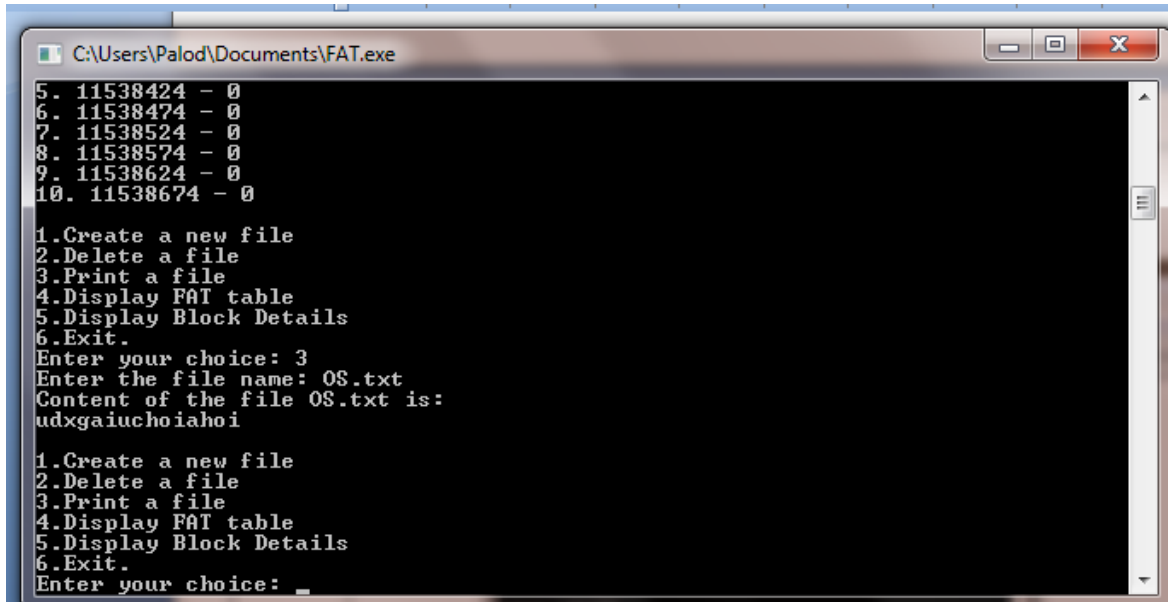
```

C:\Users\Palod\Documents\FAT.exe
2.Delete a file
3.Print a file
4.Display FAT table
5.Display Block Details
6.Exit.
Enter your choice: 5
Block address empty/free
1. 11538224 - 1
2. 11538274 - 1
3. 11538324 - 0
4. 11538374 - 0
5. 11538424 - 0
6. 11538474 - 0
7. 11538524 - 0
8. 11538574 - 0
9. 11538624 - 0
10. 11538674 - 0

1.Create a new file
2.Delete a file
3.Print a file
4.Display FAT table
5.Display Block Details
6.Exit.
Enter your choice:

```

4.TO PRINT CONTENT OF A FILE

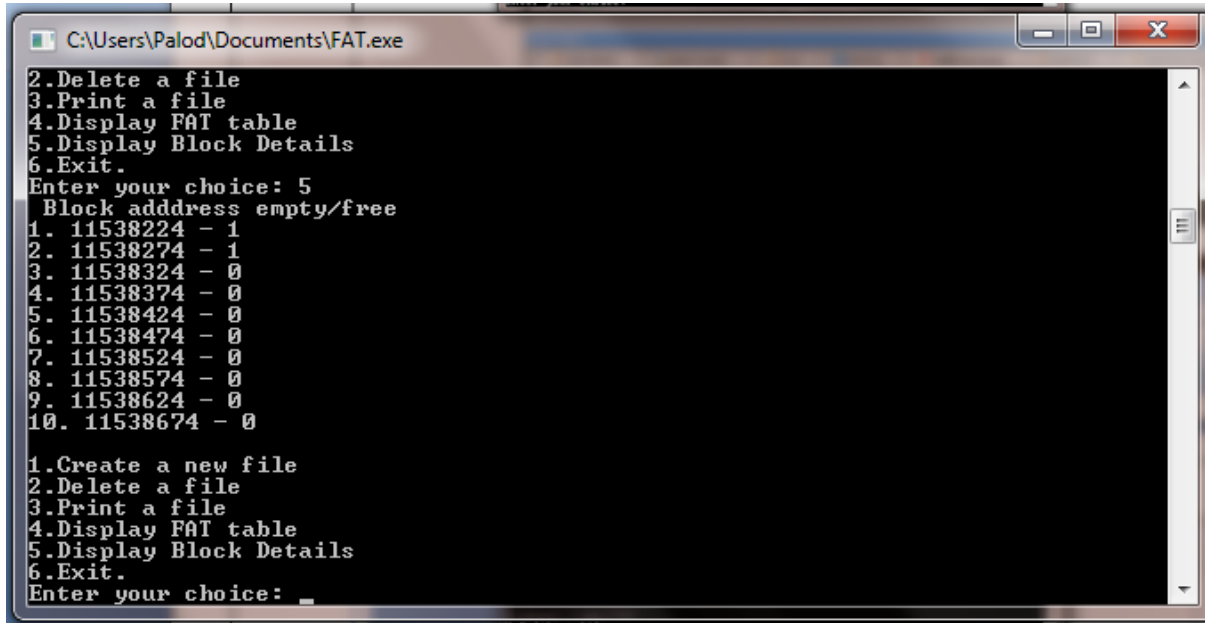


```
C:\Users\Palod\Documents\FAT.exe
5. 11538424 - 0
6. 11538474 - 0
7. 11538524 - 0
8. 11538574 - 0
9. 11538624 - 0
10. 11538674 - 0

1.Create a new file
2.Delete a file
3.Print a file
4.Display FAT table
5.Display Block Details
6.Exit.
Enter your choice: 3
Enter the file name: OS.txt
Content of the file OS.txt is:
udxgaiuchoiahoi

1.Create a new file
2.Delete a file
3.Print a file
4.Display FAT table
5.Display Block Details
6.Exit.
Enter your choice: _
```

5.TO DELETE A FILE



```
C:\Users\Palod\Documents\FAT.exe
2.Delete a file
3.Print a file
4.Display FAT table
5.Display Block Details
6.Exit.
Enter your choice: 5
Block address empty/free
1. 11538224 - 1
2. 11538274 - 1
3. 11538324 - 0
4. 11538374 - 0
5. 11538424 - 0
6. 11538474 - 0
7. 11538524 - 0
8. 11538574 - 0
9. 11538624 - 0
10. 11538674 - 0

1.Create a new file
2.Delete a file
3.Print a file
4.Display FAT table
5.Display Block Details
6.Exit.
Enter your choice: _
```


CONCLUSION

From the above comparative study between the 3 types of file allocation methods ,the best one is supposed to be the indexed file allocation method since it overcomes all the shortcomings caused by the linked allocation method and continuous allocation method.It also supports direct accessing which makes it faster and more efficient.Apart from this,we can see from the source code and outputs received that the file allocation table is a great method in order to store files and have redundancy.It makes deleting files and updating files easier .

REFERENCES

- 1.File Allocation Table. (2017, October 30). Retrieved October 31, 2017, from https://en.wikipedia.org/wiki/File_Allocation_Table
- 2.What is file allocation table (FAT) ? - Definition from WhatIs.com. (n.d.). Retrieved October 31, 2017, from <http://searchexchange.techtarget.com/definition/file-allocation-table>
- 3.What is the File Allocation Table (FAT)? - Definition from Techopedia. (n.d.). Retrieved October 31, 2017, from <https://www.techopedia.com/definition/1369/file-allocation-table-fat>
- 4.Fisher, T. (n.d.). The FAT File System: Everything You Need to Know. Retrieved October 31, 2017, from <https://www.lifewire.com/what-is-file-allocation-table-fat-2625877>

