



# VIT<sup>®</sup>

---

## Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

# OPERATING SYSTEMS

## REVIEW-3

### **Group members:**

Sahith .d (17BCE0422)

Naga Deepak bhushan .p (17BCE0791)

Saurabh Mishra (17BCI0033)

**TITLE :****DEMONSTRATION AND IMPLEMENTATION OF FILE SYSTEM****ABSTRACT:**

Every Computer requires memory. But managing that memory is important. That's why every operating system incorporates file systems in their OS. This doesn't allocate resources but when new files are created or deleted or updated all those things are managed by File Systems. File Allocation Table is one of the file systems in OS's for so long time. Here we do realise how File System works by implementing it.

**KEYWORDS:**

File Systems

File Allocation table

FAT

**INTRODUCTION :**

A file allocation table (FAT) is a file system developed for hard drives that originally used 12 or 16 bits for each cluster entry into the file allocation table. It is used by the operating system to manage files on hard drives and other computer systems. It is often also found on in flash memory, digital cameras and portable devices. It is used to store file information and extend the life of a hard drive.

In 1997 Microsoft introduced FAT32. This FAT file system increased size limits and allowed DOS real mode code to handle the format.

**TYPES OF ALLOCATION**

1. Contiguous allocation
2. Linked list allocation
3. Linked list allocation using an index .

**Contiguous Allocation:**

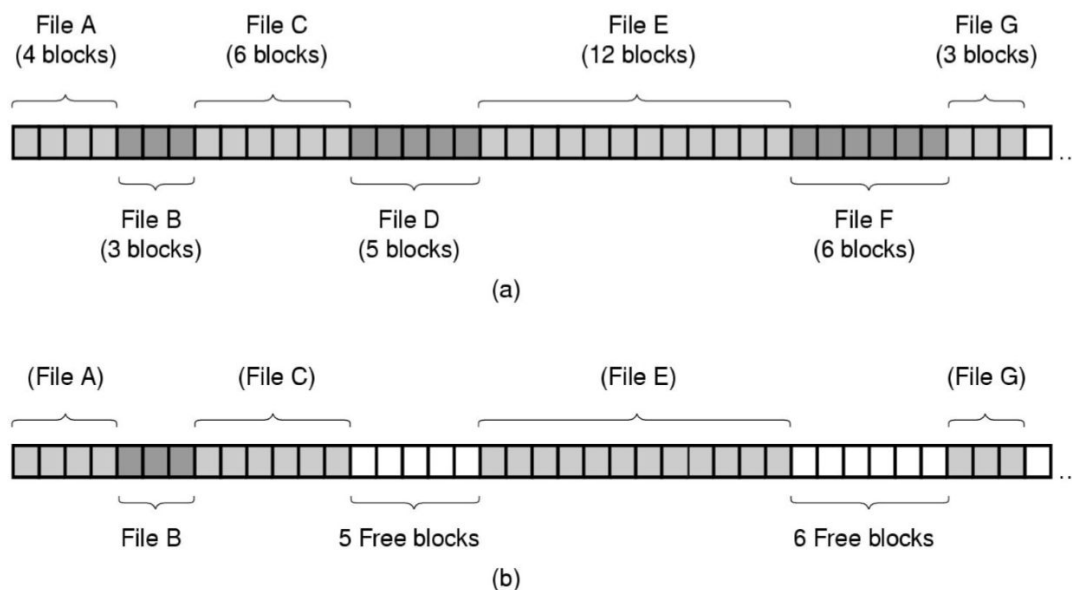
FAT (file allocation table) contains file name, start block, length.

### Advantages:

- a. Simple to implement (start block & length is enough to define a file)
- b. Fast access as blocks follow each other

### Disadvantages:

- c. Fragmentation
- d. Re-allocation (compaction)



### Structures in FAT partition

‘ **Cluster:** It is a group of sectors present in the "data area" on the FAT media. Files and directories store their data in these clusters. The size of one cluster is specified in a structure called the Boot Record and can range from a single sector to 128 sector(s).

‘ **Boot Record:** It is located within an area of reserved sectors at the very beginning of a FAT volume. BPB or BIOS Parameter Block. is what really makes up the boot record. It specifies the number of FAT copies.

‘ **File Allocation Table:** The FAT is a simple array of 12-bit, 16-bit or 32-bit data elements. The root directory can always be found immediately following the file allocation table(s).

‘ **Data Area:** It is where the contents of files and directories are stored.

‘ **Wasted Sectors**: If the number of data sectors is not evenly divisible by the cluster size you end up with a few wasted data sectors.

## **WORKING METHODOLOGY:**

Here We do FAT by contiguous allocation of memory.

### **Pseudocode:**

create buffer of size 1000;

char \*buffer= (char\*)malloc(1000); // creates a buffer size of 1000 bytes to store files

void createfile() {

    //----- Code That make entry into FAT after proper checking

    read filename

    check size

    if (size of file < mem\_available)

        {

            make enter into FAT

            blocksize\_required=size\_of\_file%100;

            update FAT with block details

        }

    }

void readfile()

{

    read filename;

    if(filename in FAT)

        parse filename;

        printf(data\_parsed);

    }

```
void deletefile(){  
    //--- delete entries in FAT  
}
```

### **SOURCE CODE:**

```
#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
  
char *buffer; //Memory created-5000 bytes  
  
int s_b[100][2],empty[100];  
  
//Stores size and number of blocks  
  
int mem=5000;  
  
//keeps track of free memory  
  
char *block[100];  
  
//store address of the blocks  
  
int *address[100];  
  
//stores address of files  
  
char name[100][30];  
  
//stores name of the file  
  
void create()  
  
//to create a file  
  
{  
  
char fname[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;
int bnum=size/100;
if(size%100!=0)
bnum=bnum+1;
s_b[n][2]=bnum;
mem=mem-(bnum*100);

```

```
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!=100)
{
*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");
}

//=====

int filenum(char fname[30])
{
int i=1,fnum=0;
while(name[i])
{
if(strcmp(name[i], fname) == 0)
{
fnum=i;
break;
}
i++;
}
return fnum;
}

//=====

void blocks(){
int i;
printf(" Block address    full/free\n");
for(i=1;i<=100;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<=100;i++)

{

if(address[i]!=0)

printf("%s %d %d\n",name[i],s_b[i][1],address[i]);

}

printf("\n");

}

//=====

void print()

{

char fname[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<100;k++)

{

printf("%c",*prt);

prt++;

}

}

printf("\n");

printf("\n");

}

else

printf("File not available:/n");

}

//=====

void rmv()

{

char fname[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*100;

for(j=0;j<b;j++)

{

int s=*(temp+j);

empty[s]=0;

}

address[fnum]=0;

}

printf("\n");

}

int main()

{

    buffer=(char*) malloc (10000);

int choice,i;

char *temp;

if (buffer == NULL)

{

fputs ("Memory error",stderr);

exit (2);

```

```

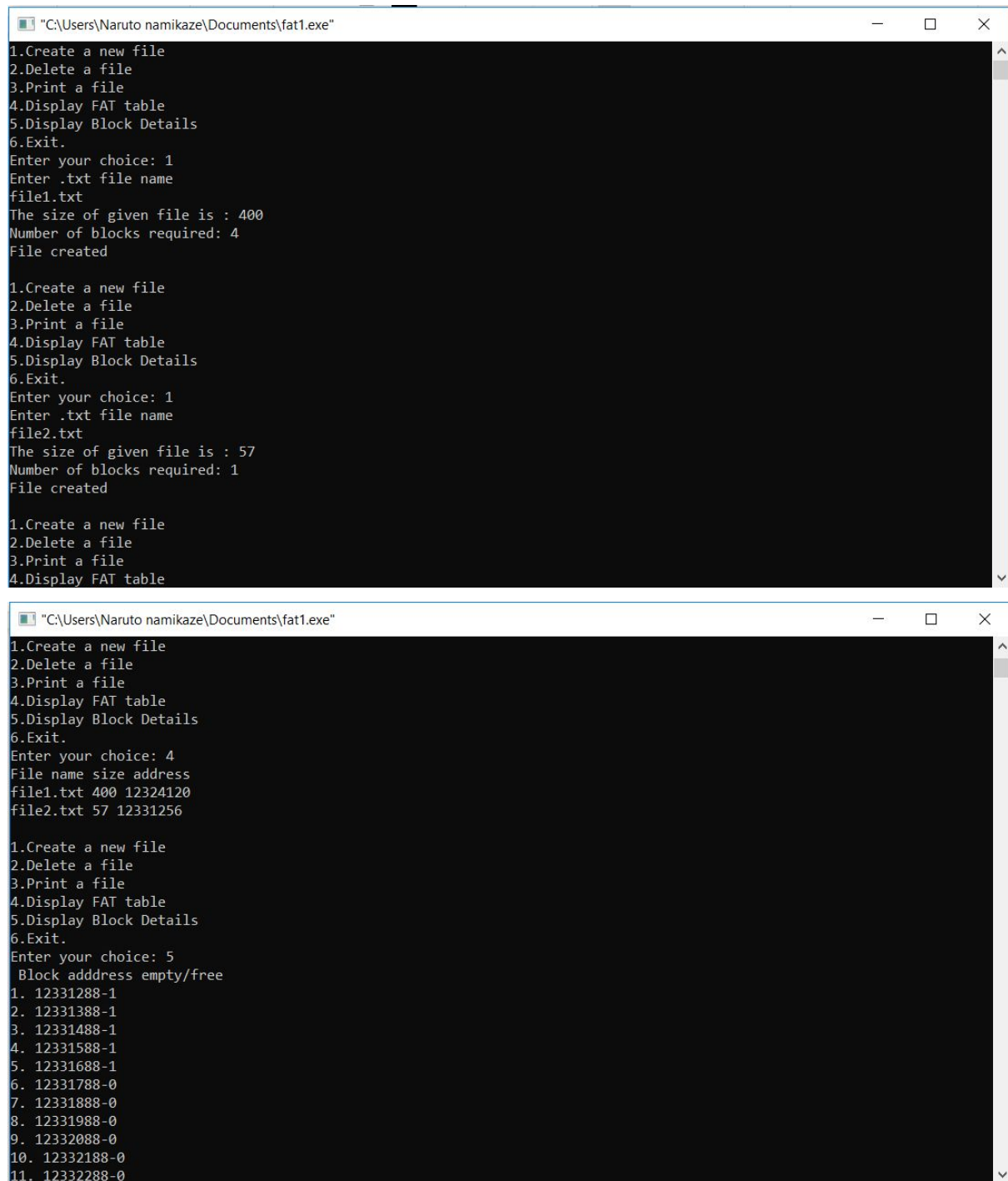
}

temp=buffer;
block[1]=buffer;
empty[1]=0;
for(i=2;i<=100;i++)
{
block[i]=block[i-1]+100;
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:
rmv();

```

```
break;  
case 3:  
    print();  
    break;  
case 4:  
    file();  
    break;  
case 5:  
    blocks();  
    break;  
case 6:  
    exit(1);  
}  
}  
return 0;  
}
```

## RESULT :



```
"C:\Users\Naruto namikaze\Documents\fat1.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
file1.txt
The size of given file is : 400
Number of blocks required: 4
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: 1
Enter .txt file name
file2.txt
The size of given file is : 57
Number of blocks required: 1
File created

1.Create a new file
2.Delete a file
3.Print a file
4.Display FAT table

"C:\Users\Naruto namikaze\Documents\fat1.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: 4
File name size address
file1.txt 400 12324120
file2.txt 57 12331256

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
Block address empty/free
1. 12331288-1
2. 12331388-1
3. 12331488-1
4. 12331588-1
5. 12331688-1
6. 12331788-0
7. 12331888-0
8. 12331988-0
9. 12332088-0
10. 12332188-0
11. 12332288-0
```

## CONCLUSION :

We realised how File Systems works and its importance and application in a hard disk, flash drive by implementing its prototype manually.