UNIVERSITY OF ENGINEERING AND TECHNOLOGY, TAXILA

FACULTY OF TELECOMMUNICATION AND INFORMATION ENGINEERING

COMPUTER ENGINEERING DEPARTMENT

Operating systems

File Allocation Strategies-2

LAB MANUAL 8

Date:	
Name:	
Reg#:	**Group:**
Marks:	**Signature:**

Operating systems

NAME OF EXPERIMENT: Simulate file Allocation strategies:

Indexed

AIM: Simulate the file allocation strategies using file allocation methods

HARDWARE REQUIREMENTS: Intel based Desktop Pc

RAM of 512 MB

SOFTWARE REQUIREMENTS:

Turbo C/ Borland C.

THEORY:

Indexed allocation

In linked allocation it is difficult to maintain FAT – so instead of that method indexed allocation method is used. Indexed allocation method solves all the problems in the linked allocation by bringing all the pointers together into one location called index block.

ALGORITHM:

1. Start
2. Read the number of files
3. Read the index block for each file.
4. For each file, read the number of blocks occupied and number of blocks of the file.
5. Link all the blocks of the file to the index block.
6. Display the file name, index block , and the blocks occupied by the file.
7. stop

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
main()
{
int n,m[20],i,j,ib[20],b[20][20];
clrscr();
printf("Enter no. of files:");
scanf("%d",&n);
```

```
for(i=0;i<n;i++)
{ printf("Enter index block :",i+1);
scanf("%d",&ib[i]);
printf("Enter blocks occupied by file%d:",i+1);
scanf("%d",&m[i]);
printf("enter blocks of file%d:",i+1);
for(j=0;j<m[i];j++)
scanf("%d",&b[i][j]);
} printf("\nFile\t index\tlength\n");
for(i=0;i<n;i++)
printf("%d\t%d\t%d\n",i+1,ib[i],m[i]);
printf("blocks occupiedare:");
 for(i=0;i<n;i++)
 { printf("fileno%d",i+1);
 for(j=0;j<m[i];j++)
 printf("\t%d--->%d\n",ib[i],b[i][j]);
 printf("\n");
 }
getch();
}
```

OUTPUT:

```
Enter no. of files:2
Enter index block 3
Enter blocks occupied by file1: 4
enter blocks of file1:9
 4 6 7
Enter index block 5
Enter blocks occupied by file2:2
enter blocks of file2: 10 8
 File index length
```

1 3 4

2 5 2

blocksoccupiedare:

file1

3--->9

3--->4

3--->6

3--->7

file2

5--->10

5--->8

Task

1. What file allocation strategy is most appropriate for random access files?

2.Define File?

3.Define Directory?

4. Why we use file allocation strategies?

5.what are the advantages and dis-advantages Indexed Allocation?

Or

INDEXED FILE ALLOCATION

AIM: Write a C Program to implement Indexed File Allocation method.

Description:

Indexed allocation:

Indexed allocation supports both sequential and direct access files. The file indexes are not physically stored as a part of the file allocation table. Whenever the file size increases, we can easily add some more blocks to the index. In this strategy, the file allocation table contains a single entry for each file. The entry consisting of one index block, the index blocks having the pointers to the other blocks. No external fragmentation.

Algorithm for Indexed File Allocation:

Step 1: Start.

Step 2: Let n be the size of the buffer

Step 3: check if there are any producer

Step 4: if yes check whether the buffer is full

Step 5: If no the producer item is stored in the buffer

Step 6: If the buffer is full the producer has to wait

Step 7: Check there is any consumer.If yes check whether the buffer is empty

Step 8: If no the consumer consumes them from the buffer

Step 9: If the buffer is empty, the consumer has to wait.

Step 10: Repeat checking for the producer and consumer till required

Step 11: Terminate the process.

```
/* Program to simulate indexed file allocation strategy */
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
void main()
{
int f[50], index[50],i, n, st, len, j, c, k, ind,count=0;
clrscr();
for(i=0;i<50;i++)
f[i]=0;
x:printf("Enter the index block: ");
scanf("%d",&ind);
if(f[ind]!=1)
{
 printf("Enter no of blocks needed and no of files for the index %d on the disk : \n",
ind);
 scanf("%d",&n);
}
else
{
 printf("%d index is already allocated \n",ind);
 goto x;
```

```
}
y: count=0;
for(i=0;i<n;i++)
{
 scanf("%d", &index[i]);
 if(f[index[i]]==0)
 count++;
}
if(count==n)
{
 for(j=0;j<n;j++)
 f[index[j]]=1;
 printf("Allocated\n");
 printf("File Indexed\n");
 for(k=0;k<n;k++)
 printf("%d-------->%d : %d\n",ind,index[k],f[index[k]]);
}
else
{
printf("File in the index is already allocated \n");
printf("Enter another file indexed");
goto y;
}
printf("Do you want to enter more file(Yes - 1/No - 0)");
scanf("%d", &c);
if(c==1)
 goto x;
else
 exit(0);
getch();
```

}

OUTPUT 1:

Enter the index block: 5

Enter no of blocks needed and no of files for the index 5 on the disk :

4

1 2 3 4

Allocated

File Indexed

5-------->1 : 1

5-------->2 : 1

5-------->3 : 1

5-------->4 : 1

Do you want to enter more file(Yes - 1/No - 0)1

Enter the index block: 4

4 index is already allocated

Enter the index block: 6

Enter no of blocks needed and no of files for the index 6 on the disk :

2

7 8

A5llocated

File Indexed

6-------->7 : 1

6-------->8 : 1

Do you want to enter more file(Yes - 1/No - 0)0

OUTPUT 2:

Enter the index block: 4

Enter no of blocks needed and no of files for the index 4 on the disk :

3

1 2 3

Allocated

File Indexed

4-------->1 : 1

4-------->2 : 1

4-------->3 : 1

Do you want to enter more file(Yes - 1/No - 0)0

//

THEORY:

- **Linked Allocation**

Linked allocation of disk space overcomes all the problems of contiguous allocation. In linked allocation each file is a linked list of disk blocks where the disk blocks may be scattered anywhere on the disk. The directory contains a pointer to the first and last blocks of the file.

Disadvantages: Space required maintaining pointers.

ALGORITHM:

1. Start
2. Read the number of files
3. For each file, read the file name, starting block, number of blocks and block numbers of the file.
4. Start from the starting block and link each block of the file to the next block in a linked list fashion.

5. Display the file name, starting block, size of the file , and the blocks occupied by the file.
6. stop.

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
struct file
{
char fname[10];
int start,size,block[10];
}f[10];
```

Operating systems

```
main()
{
int i,j,n;
clrscr();
printf("Enter no. of files:");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("Enter file name:");
scanf("%s",&f[i].fname);
printf("Enter starting block:");
scanf("%d",&f[i].start);
f[i].block[0]=f[i].start;
printf("Enter no.of blocks:");
scanf("%d",&f[i].size);
printf("Enter block numbers:");
for(j=1;j<=f[i].size;j++)
{
scanf("%d",&f[i].block[j]);
}
}
printf("File\tstart\tsize\tblock\n");
for(i=0;i<n;i++)
{
printf("%s\t%d\t%d\t",f[i].fname,f[i].start,f[i].size);
for(j=0;j<f[i].size;j++)
printf("%d--->",f[i].block[j]);
printf("%d",f[i].block[j]);
printf("\n");
}
```

```
getch();
}
```

OUTPUT:

```
Enter no. of files:2
Enter file name:venkat
Enter starting block:20
Enter no.of blocks:6
Enter block numbers: 4
12
15
45
32
25
Enter file name:rajesh
Enter starting block:12
Enter no.of blocks:5
Enter block numbers:6
5
4
3
2
File start size block
venkat 20 6 20--->4--->12--->15--->45--->32--->25
rajesh 12 5 12--->6--->5--->4--->3--->2
```

Task

1.What file access pattern is particularly suited to chained file allocation on disk?

2. What file allocation strategy is most appropriate for random access files?

3.Mention different file allocation strategies?

4. Why we use file allocation strategies?

5.what are the advantages and dis-advantages of each strategies?

Or

LINKED FILE ALLOCATION

AIM: Write a C Program to implement Linked File Allocation method.

Description:

Linked allocation:

It is easy to allocate the files, because allocation is on an individual block basis. Each block contains a pointer to the next free block in the chain. Here also the file allocation table consisting of a single entry for each file. Using this strategy any free block can be added to a chain very easily. There is a link between one block to another block, that's why it is said to be linked allocation. We can avoid the external fragmentation.

Algorithm for Linked File Allocation:

Step 1: Create a queue to hold all pages in memory

Step 2: When the page is required replace the page at the head of the queue

Step 3: Now the new page is inserted at the tail of the queue

Step 4: Create a stack

Step 5: When the page fault occurs replace page present at the bottom of the stack

Step 6: Stop the allocation.

```
/* Program to simulate linked file allocation strategy */
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
void main()
{
int f[50], p,i, st, len, j, c, k, a;
clrscr();
for(i=0;i<50;i++)
f[i]=0;
printf("Enter how many blocks already allocated: ");
scanf("%d",&p);
printf("Enter blocks already allocated: ");
```

```
for(i=0;i<p;i++)
{
 scanf("%d",&a);
 f[a]=1;
}
x: printf("Enter index starting block and length: ");
scanf("%d%d", &st,&len);
k=len;
if(f[st]==0)
{
  for(j=st;j<(st+k);j++)
 {
 if(f[j]==0)
 {
 f[j]=1;
 printf("%d-------->%d\n",j,f[j]);
}
 else
 {
 printf("%d Block is already allocated \n",j);
 k++;
 }
 }
}
else
printf("%d starting block is already allocated \n",st);
printf("Do you want to enter more file(Yes - 1/No - 0)");
scanf("%d", &c);
if(c==1)
 goto x;
```

```
else
 exit(0);
getch();
}
```

OUTPUT 1:

Enter how many blocks already allocated: 3

Enter blocks already allocated: 1 3 5

Enter index starting block and length: 2 2

2-------->1

3 Block is already allocated

4-------->1

Do you want to enter mre file(Yes - 1/No - 0)0

OUTPUT 2:

Enter blocks already allocated: 2 4 6 8 10 12

Enter index starting block and length: 3 10

3-------->1

4 Block is already allocated

5-------->1

6 Block is already allocated

7-------->1

8 Block is already allocated

9-------->1

10 Block is already allocated

11-------->1

12 Block is already allocated

13-------->1

14-------->1

15-------->1

16-------->1

17-------->1

Do you want to enter more file(Yes - 1/No - 0)1

Operating systems

Enter index starting block and length: 5 1

5 starting block is already allocated

Do you want to enter more file(Yes - 1/No - 0)1

Enter index starting block and length: 18 2

18-------->1

19-------->1

Do you want to enter more file(Yes - 1/No - 0)0