

Experiment No 10

Class: SE Comp

Year: 2020-21

Performed by: Danyl Fernandes, 72

Aim: Write a program to simulate File allocation strategies typically sequential, indexed and linked files.

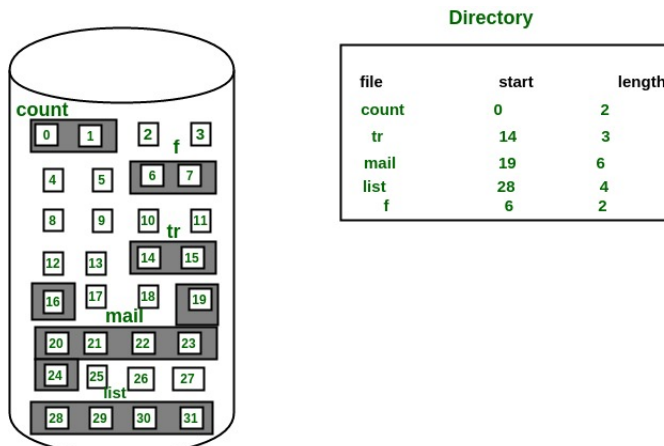
Theory:

File Allocation Methods:

- The allocation methods define how the files are stored in the disk blocks. There are three main disk space or file allocation methods.
 - Contiguous Allocation
 - Linked Allocation
 - Indexed Allocation
- They provide efficient disk space utilization.
- They help in fast access to the file blocks.

Contiguous Allocation:

- In this scheme, each file occupies a contiguous set of blocks on the disk.
- For example, if a file requires n blocks and is given a block b as the starting location, then the blocks assigned to the file will be: $b, b+1, b+2, \dots, b+n-1$.
- This means that given the starting block address and the length of the file (in terms of blocks required), we can determine the blocks occupied by the file.
- The directory entry for a file with contiguous allocation contains
 - Address of starting block
 - Length of the allocated portion.
- The file 'mail' in the following figure starts from the block 19 with length = 6 blocks.
- Therefore, it occupies 19, 20, 21, 22, 23, 24 blocks.



Advantages:

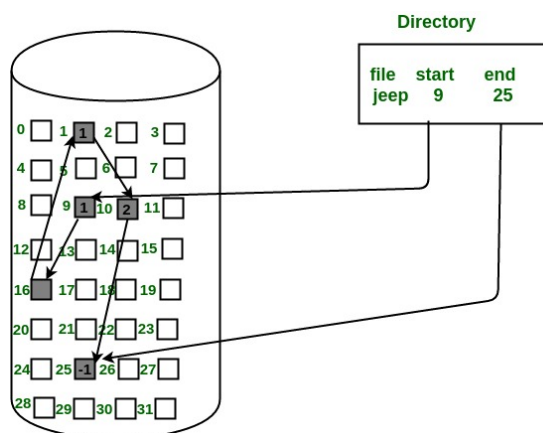
- Both the Sequential and Direct Accesses are supported by this.
- For direct access, the address of the kth block of the file which starts at block b can easily be obtained as (b+k).
- This is extremely fast since the number of seeks are minimal because of contiguous allocation of file blocks.

Disadvantages:

- This method suffers from both internal and external fragmentation. This makes it inefficient in terms of memory utilization.
- Increasing file size is difficult because it depends on the availability of contiguous memory at a particular instance.

Linked List Allocation:

- In this scheme, each file is a linked list of disk blocks which need not be contiguous. The disk blocks can be scattered anywhere on the disk.
- The directory entry contains a pointer to the starting and the ending file block. Each block contains a pointer to the next block occupied by the file.
- The file 'jeep' in the following image shows how the blocks are randomly distributed. The last block (25) contains -1 indicating a null pointer and does not point to any other block.



Advantages:

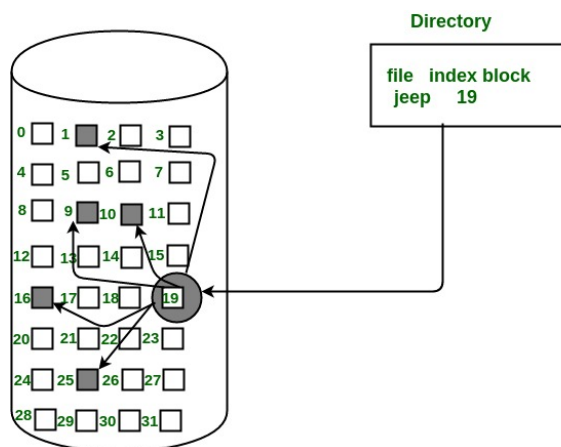
- This is very flexible in terms of file size. File size can be increased easily since the system does not have to look for a contiguous chunk of memory.
- This method does not suffer from external fragmentation. This makes it relatively better in terms of memory utilization.

Disadvantages:

- Because the file blocks are distributed randomly on the disk, a large number of seeks are needed to access every block individually. This makes linked allocation slower.
- It does not support random or direct access. We can not directly access the blocks of a file. A block k of a file can be accessed by traversing k blocks sequentially (sequential access) from the starting block of the file via block pointers.
- Pointers required in the linked allocation incur some extra overhead.

Indexed Allocation

- In this scheme, a special block known as the Index block contains the pointers to all the blocks occupied by a file.
- Each file has its own index block. The ith entry in the index block contains the disk address of the ith file block.
- The directory entry contains the address of the index block as shown in the image



Advantages:

- This supports direct access to the blocks occupied by the file and therefore provides fast access to the file blocks.
- It overcomes the problem of external fragmentation.

Disadvantages:

- The pointer overhead for indexed allocation is greater than linked allocation.
- For very small files, say files that expand only 2-3 blocks, the indexed allocation would keep one entire block (index block) for the pointers which is inefficient in terms of memory utilization.
- However, in linked allocation we lose the space of only 1 pointer per block.

Large-block Mechanisms:

- **Linked scheme:**
 - This scheme links two or more index blocks together for holding the pointers.
 - Every index block would then contain a pointer or the address to the next index block.
- **Multilevel index:**
 - In this policy, a first level index block is used to point to the second level index blocks which in turn points to the disk blocks occupied by the file.
 - This can be extended to 3 or more levels depending on the maximum file size.
- **Combined Scheme:**
 - In this scheme, a special block called the Inode (information Node) contains all the information about the file such as the name, size, authority, etc and the remaining space of Inode is used to store the Disk Block addresses which contain the actual file.
 - The first few of these pointers in Inode point to the direct blocks i.e the pointers contain the addresses of the disk blocks that contain data of the file.
 - The next few pointers point to indirect blocks. Indirect blocks may be single indirect, double indirect or triple indirect.
 - Single Indirect block is the disk block that does not contain the file data but the disk address of the blocks that contain the file data.
 - Similarly, double indirect blocks do not contain the file data but the disk address of the blocks that contain the address of the blocks containing the file data.

Conclusion:

- In this experiment, we were successfully able to simulate File allocation strategies typically sequential, indexed and linked files.

Implementation:

Sequential Allocation:

```
#include <iostream>
#include <conio.h>
using namespace std;

void recurse(int files[]){
    int flag = 0, startBlock, len, k;
    cout << "Enter the starting block and the length of the
files: ";
    cin >> startBlock >> len;
    for (int j=startBlock; j<(startBlock+len); j++){
        if (files[j] == 0)
            flag++;
    }
    if(len == flag){
        for (int k=startBlock; k<(startBlock+len); k++){
            if (files[k] == 0){
                files[k] = 1;
                cout << k <<"\t" << files[k] << endl;
            }
        }
        if (k != (startBlock+len-1))
            cout << "The file is allocated to the disk" << endl;
    }
    else
        cout << "The file is not allocated to the disk" << endl;

    cout << "Do you want to enter more files?" << endl;
    int ch;
    cout << "Press 1 for YES, 0 for NO: ";
    cin >> ch;
    if (ch == 1)
        recurse(files);
    else
        exit(0);
    return;
}
```

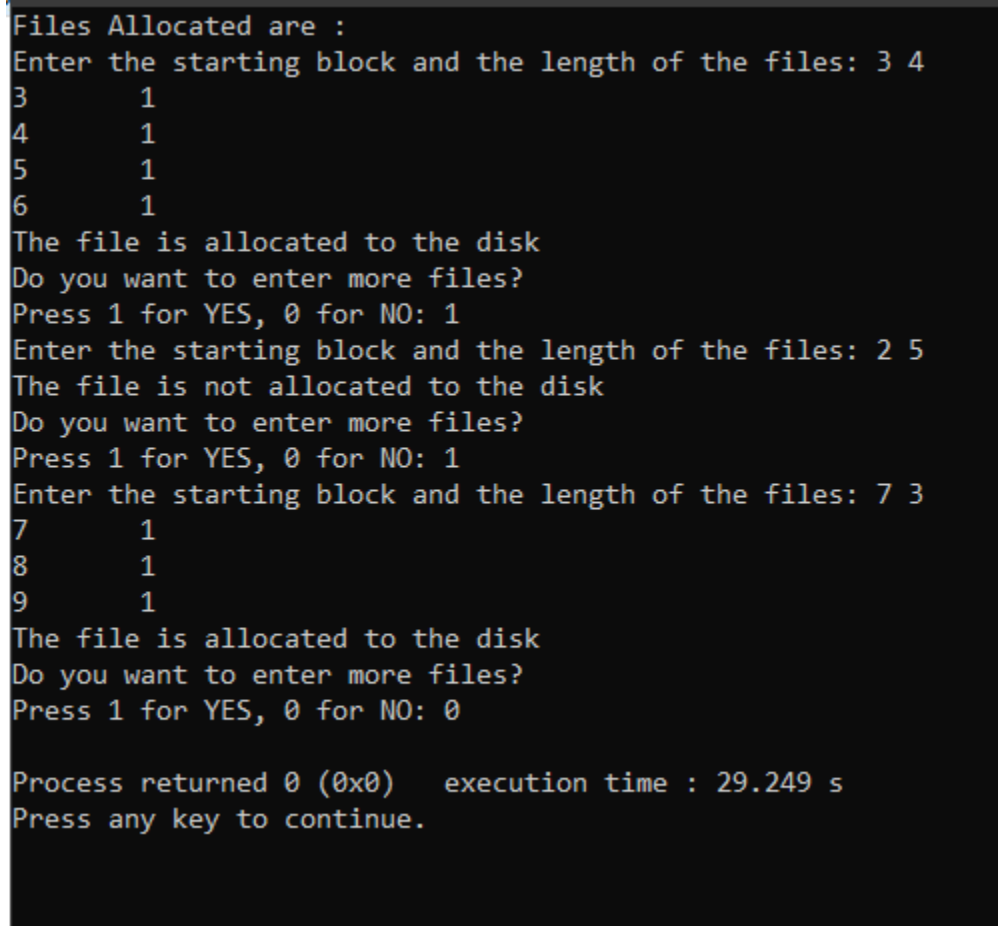
```

int main() {
    int files[50];
    for(int i=0;i<50;i++)
        files[i]=0;
    cout << "Files Allocated are :" << endl;

    recurse(files);
    getch();
    return 0;
}

```

Output:



```

Files Allocated are :
Enter the starting block and the length of the files: 3 4
3      1
4      1
5      1
6      1
The file is allocated to the disk
Do you want to enter more files?
Press 1 for YES, 0 for NO: 1
Enter the starting block and the length of the files: 2 5
The file is not allocated to the disk
Do you want to enter more files?
Press 1 for YES, 0 for NO: 1
Enter the starting block and the length of the files: 7 3
7      1
8      1
9      1
The file is allocated to the disk
Do you want to enter more files?
Press 1 for YES, 0 for NO: 0

Process returned 0 (0x0)   execution time : 29.249 s
Press any key to continue.

```

Linked Allocation:

```
#include <iostream>
#include <conio.h>
using namespace std;

void recursivePart(int pages[]){
    int st, len;
    cout << "Enter the index of the starting block and its
length: ";
    cin >> st >> len;
    int k = len;
    if (pages[st] == 0){
        for (int j = st; j < (st + k); j++){
            if (pages[j] == 0){
                pages[j] = 1;
                cout << j << "----->" << pages[j] << endl;
            }
            else {
                cout << "The block " << j << " is already
allocated" << endl;
                k++;
            }
        }
    }
    else
        cout << "The block " << st << " is already allocated" <<
endl;
    cout << "Do you want to enter more files?" << endl;
    cout << "Enter 1 for Yes, Enter 0 for No: ";
    int c;
    cin >> c;
    if (c==1)
        recursivePart(pages);
    else
        exit(0);
    return;
}
```

```

int main(){
    int pages[50], p, a;

    for (int i = 0; i < 50; i++)
        pages[i] = 0;
    cout << "Enter the number of blocks already allocated: ";
    cin >> p;
    cout << "Enter the blocks already allocated: ";
    for (int i = 0; i < p; i++){
        cin >> a;
        pages[a] = 1;
    }

    recursivePart(pages);
    getch();
    return 0;
}

```

Output:

```

Enter the number of blocks already allocated: 2
Enter the blocks already allocated: 3
4
Enter the index of the starting block and its length: 1 6
1----->1
2----->1
The block 3 is already allocated
The block 4 is already allocated
5----->1
6----->1
7----->1
8----->1
Do you want to enter more files?
Enter 1 for Yes, Enter 0 for No: 1
Enter the index of the starting block and its length: 6 2
The block 6 is already allocated
Do you want to enter more files?
Enter 1 for Yes, Enter 0 for No: 1
Enter the index of the starting block and its length: 3 2
The block 3 is already allocated
Do you want to enter more files?
Enter 1 for Yes, Enter 0 for No: 0

Process returned 0 (0x0)   execution time : 35.195 s
Press any key to continue.

```


Indexed Allocation:

```
#include <iostream>
#include <conio.h>
#include <stdlib.h>

using namespace std;

int files[50], indexBlock[50], indBlock, n;
void recurse1();
void recurse2();

void recurse1(){
    cout << "Enter the index block: ";
    cin >> indBlock;
    if (files[indBlock] != 1){
        cout << "Enter the number of blocks and the number of
files needed for the index " << indBlock << " on the disk: ";
        cin >> n;
    }
    else{
        cout << indBlock << " is already allocated" << endl;
        recurse1();
    }
    recurse2();
}

void recurse2(){
    int flag = 0;
    for (int i=0; i<n; i++){
        cin >> indexBlock[i];
        if (files[indexBlock[i]] == 0)
            flag++;
    }
    if (flag == n){
        for (int j=0; j<n; j++){
            files[indexBlock[j]] = 1;
        }
        cout << "Allocated" << endl;
        cout << "File Indexed" << endl;
        for (int k=0; k<n; k++){
```

```

        cout << indBlock << " -----> " << indexBlock[k] <<
": " << files[indexBlock[k]] << endl;
    }
}
else{
    cout << "File in the index is already allocated" <<
endl;
    cout << "Enter another indexed file" << endl;
    recurse2();
}
cout << "Do you want to enter more files?" << endl;
cout << "Enter 1 for Yes, Enter 0 for No: ";
int ch;
cin >> ch;
if (ch == 1)
    recurse1();
else
    exit(0);
return;
}

int main() {
    for(int i=0;i<50;i++)
        files[i]=0;

    recurse1();
    return 0;
}

```

Output:

```

Enter the index block: 4
Enter the number of blocks and the number of files needed for the index 4 on the disk: 3 4 2 3
Allocated
File Indexed
4 -----> 4: 1
4 -----> 2: 1
4 -----> 3: 1
Do you want to enter more files?
Enter 1 for Yes, Enter 0 for No: 0

Process returned 0 (0x0)   execution time : 12.104 s
Press any key to continue.

```