Koc University

Spring2020

COMP304 – Operating Systems

Project 3: Space Allocation Methods

Arda Duman 60609

Project Aim:

In this project, we are asked to implement 2 different allocation types: Contiguous and linked allocation. We are asked to implement them by creating a Directory Table and File Allocation Table. In this project, all of the parts are fully working in my implementation. Further information is inside the code. Every line is explained by my comments.

To run the code:

We are given a set of inputs with different block sizes and contents. In my code there are two seperated main classes for Contiguous and Linked Allocation. For example, to run Linked Allocation for an input file, you need to run the main class of it, which is named the same.

In the code there is a FILE_NAME parameter. You need to assign it in the code with the name of the input file that you want to test. Also there is a BLOCK_SIZE parameter which should be the first number in the name of the file. You need to update it before running.

In my implementation, there are 5 classes which are :

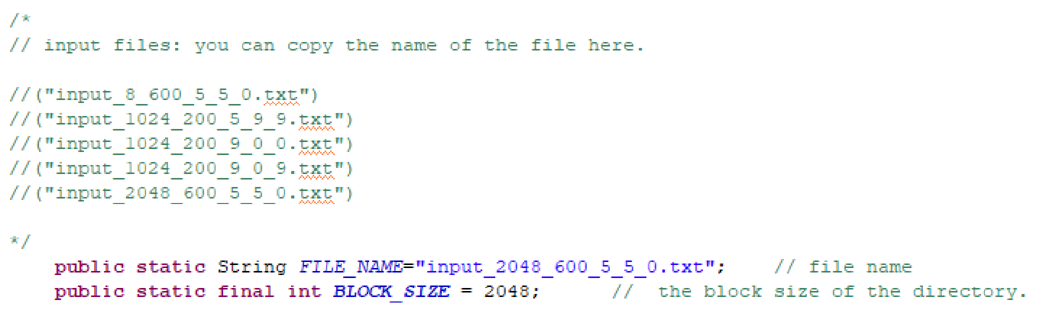
ContiguousAllocation: it is the main class of the contiguous allocation.

LinkedAllocation: it is the main class of the linked allocation.

Block: it is the content of the directory.

DTContent: it is the content of the DT.

Pair: a pair class is used for creatin a <start, size> pair for FAT.

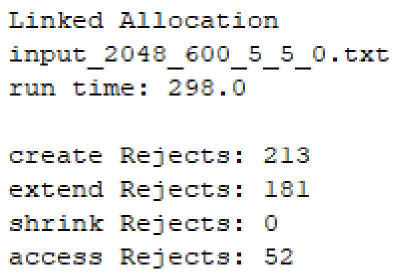


```
/*
// input files: you can copy the name of the file here.

//("input_8_600_5_5_0.txt")
//("input_1024_200_5_9_9.txt")
//("input_1024_200_9_0_0.txt")
//("input_1024_200_9_0_9.txt")
//("input_2048_600_5_5_0.txt")

*/
    public static String FILE_NAME="input_2048_600_5_5_0.txt";    // file name
    public static final int BLOCK_SIZE = 2048;      //  the block size of the directory.
```

After you run the code the output should look like this:



```
Linked Allocation
input_2048_600_5_5_0.txt
run time: 298.0

create Rejects: 213
extend Rejects: 181
shrink Rejects: 0
access Rejects: 52
```

Experimentation and Analysis:

According to my experiments, the rejected requests for different input files are below:

input_8_600_5_5_0.txt:

create Rejects: 215

extend Rejects: 201

shrink Rejects: 0

access Rejects: 39

----

input_1024_200_5_9_9.txt :

create Rejects: 18

extend Rejects: 201

shrink Rejects: 104

access Rejects: 17

----

input_1024_200_9_0_0.txt :

create Rejects: 80

extend Rejects: 0

shrink Rejects: 0

access Rejects: 42115

----

input_1024_200_9_0_9.txt :

create Rejects: 0

extend Rejects: 0

shrink Rejects: 0

access Rejects: 0

----

input_2048_600_5_5_0.txt :

create Rejects: 213

extend Rejects: 181

shrink Rejects: 0

access Rejects: 52

For an input file the result is the same for both Linked and Contiguous Allocation.

Average Running Times

input_8_600_5_5_0.txt:

For contiguous allocation 5 run times:

```
run time 1: 137.0

run time 2: 125.0

run time 3: 127.0

run time 4: 138.0

run time 5: 128.0

Average: 131.0 Milliseconds
```

For linked allocation 5 run times:

```
run time 1: 271.0

run time 2: 303.0

run time 3: 319.0

run time 4: 300.0

run time 5: 279.0

Average: 294.4 Milliseconds
```

input_1024_200_5_9_9.txt:

For contiguous allocation 5 run times:

```
run time 1: 275.0

run time 2: 225.0

run time 3: 201.0

run time 4: 215.0

run time 5: 218.0

Average: 226.8 Milliseconds
```

For linked allocation 5 run times:

```
run time 1: 500.0

run time 2: 490.0

run time 3: 531.0

run time 4: 534.0

run time 5: 516.0

Average: 514.2 Milliseconds
```

input_1024_200_9_0_0.txt:

For contiguous allocation 5 run times:

```
run time 1: 610.0

run time 2: 588.0

run time 3: 611.0

run time 4: 600.0

run time 5: 564.0

Average: 594,6.0 Milliseconds
```

For linked allocation 5 run times:

```
run time 1: 1295.0

run time 2: 1273.0

run time 3: 1301.0

run time 4: 1357.0

run time 5: 1293.0

Average: 1303.8 Milliseconds
```

input_1024_200_9_0_9.txt :

For contiguous allocation 5 run times:

```
run time 1: 121.0

run time 2: 107.0

run time 3: 101.0

run time 4: 102.0

run time 5: 105.0
```

Average: 107.2 Milliseconds

For linked allocation 5 run times:

```
run time 1: 125.0

run time 2: 124.0

run time 3: 134.0

run time 4: 146.0

run time 5: 131.0
```

Average: 131.0 Milliseconds

input_2048_600_5_5_0.txt:

For contiguous allocation 5 run times:

```
run time 1: 128.0

run time 2: 131.0

run time 3: 128.0

run time 4: 135.0

run time 5: 146.0
```

Average: 133.6 Milliseconds

For linked allocation 5 run times:

```
run time 1: 283.0

run time 2: 304.0

run time 3: 283.0

run time 4: 280.0

run time 5: 287.0
```

Average: 287.4 Milliseconds

Questions:

Q1: With test instances having a block size of 1024, in which cases (inputs) contiguous allocation has a shorter average operation time? Why? What are the dominating operations in these cases? In which linked is better, why?

- In my all tests for the inputs with block size 1024, contiguous allocation has a shorter average operation time. However, for the input “input_1024_200_9_0_0.txt” the average time for contiguous is less than the linked allocation. This is the case because, in this input file the dominating operation is access and contiguous allocation is expected to have a shorter run time. In my all tests, contiguous is faster. However, I expected to see a faster linked allocation result in “input_1024_200_5_9_9.txt” because in this file there are a lot of extend, create, and shrink operations, which should be faster in linked allocation.

Q2: Comparing the difference between the creation rejection ratios with block size 2048 and 8, what can you conclude? How did dealing with smaller block sizes affect the FAT memory utilization?

- For block size 2048, in the file there are 600 create requests and 213 of them are rejected. So, the creation rejection ratio is 35,5%.

For block size 8, in the file there are 600 create requests and 215 of them are rejected. So, the creation rejection ratio is 35,83%.

These ratios are the same for both linked and contiguous allocation because, I give an extra memory for FAT. However, if we allocate the memory for the FAT and not give an extra space, the block size will affect the ratio between DT and FAT. Bigger block sizes will need more memory for FAT and it would increase the rejection ratio.

Q3: FAT is a popular way to implement linked allocation strategy. This is because it permits faster access compared to the case where the pointer to the next block is stored as a part of the concerned block. Explain why this provides better space utilization

- If we store the pointer for the next block in the directory block itself, it will decrease the space we can allocate in a single block. Using FAT, we use all blocks fully and we can iterate over FAT to access a block instead of iterating over the whole directory, which is a faster operation.

Q4: If you have extra memory available of a size equal to the size of the DT, how can this improve the performance of your defragmentation?

- We could use the extra space to store the blocks to be shifted for the extend operation. However, in our case we shift the files block by block and it is slower than shifting the whole file at once.

Q5: How much, at minimum, extra memory do you need to guarantee reduction in the number of rejected extensions in the case of contiguous allocations?

- In our implementation, all extension rejections are due to lack of free spaces in the directory. If we have enough space for the extension operations, it will decrease the amount of the extension rejections.

Inside my submission you can find:

- 5 .java files: ContiguousAllocation.java, LinkedAllocation.java, Block.java, DTContent.java, Pair.java