2021 2-1 I233E Operating System Assignment#2

Introduction

1.Bitmap

Bitmap is an array consist of 32-bit unsigned integers. Each bit represent the state of a block in disk.

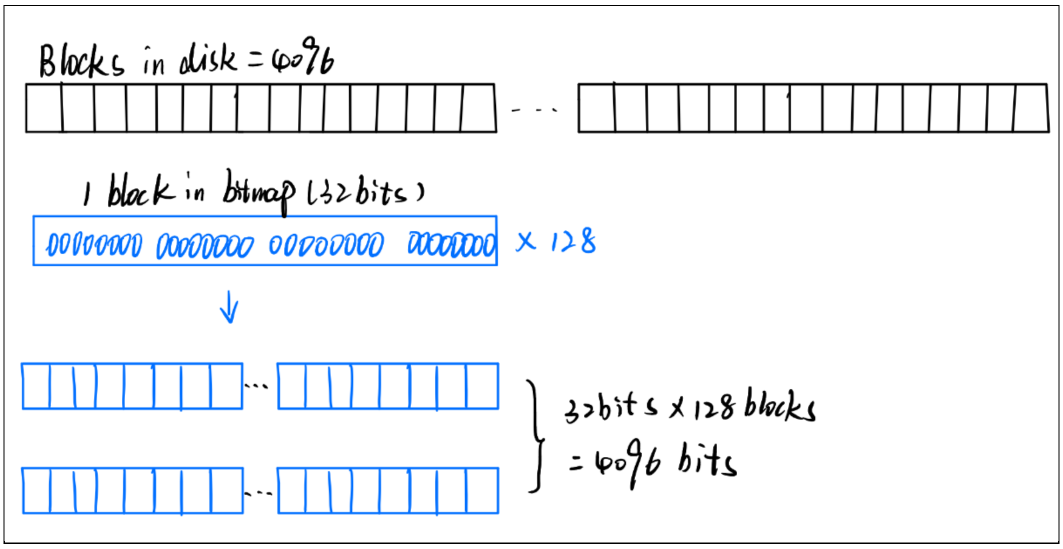


Figure1. Bitmap

2. File block table

File block table is the abstract representation of linked list of file blocks. It is also an array of integers. a non-zero integer implies that the corresponding block in disk is being used. Also, the pointer to the next linked block is recorded. "-1" means this is the tail of a file block linked list.

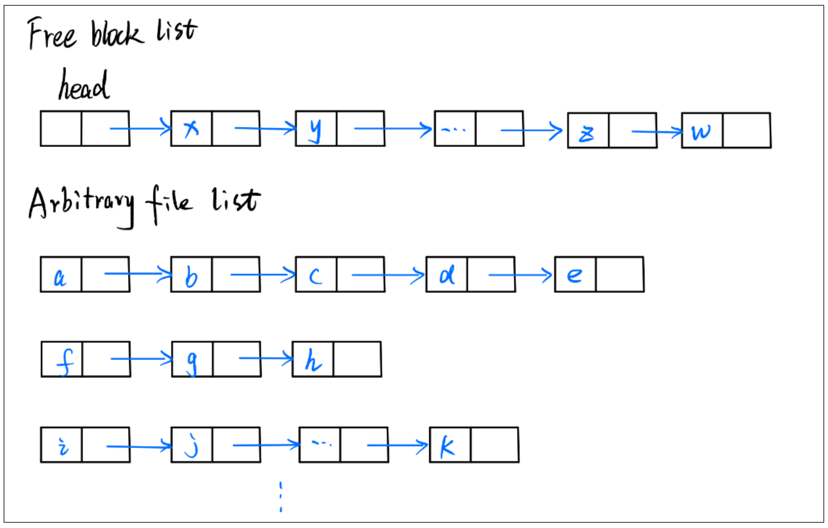


Figure2. fbt

Initialization

- Free space management(set all block as free block)
- (option) Super block(parameter definition)(# of free blocks, # of directory entries, and so on)

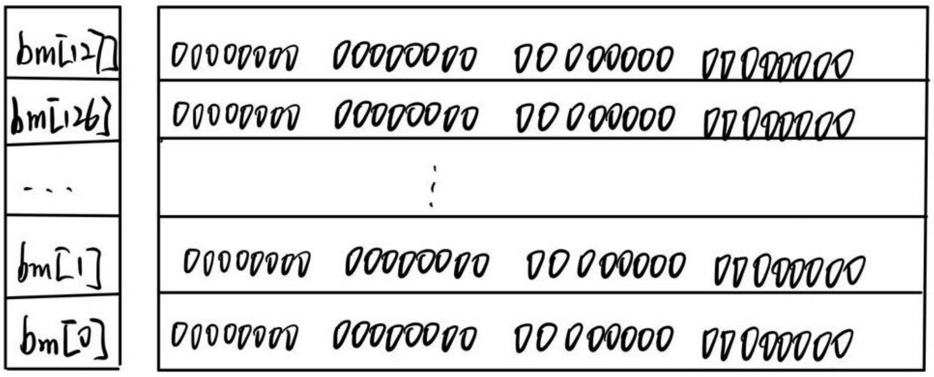


Figure3. Bitmap initialization

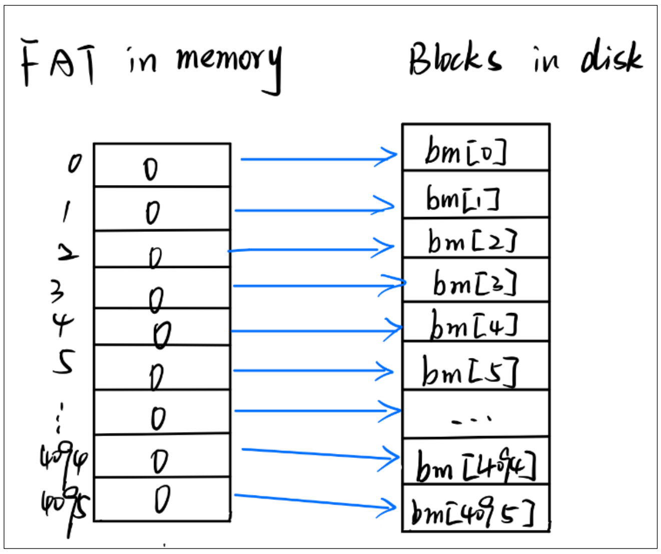


Figure4. FAT initialization

Allocation

- Determine whether there are n consecutive blocks
- in fbt, the file blocks will be allocated properly if there are enough blocks. Because there is no need to allocate consecutive n blocks of free space in advance.
- update # of free blocks

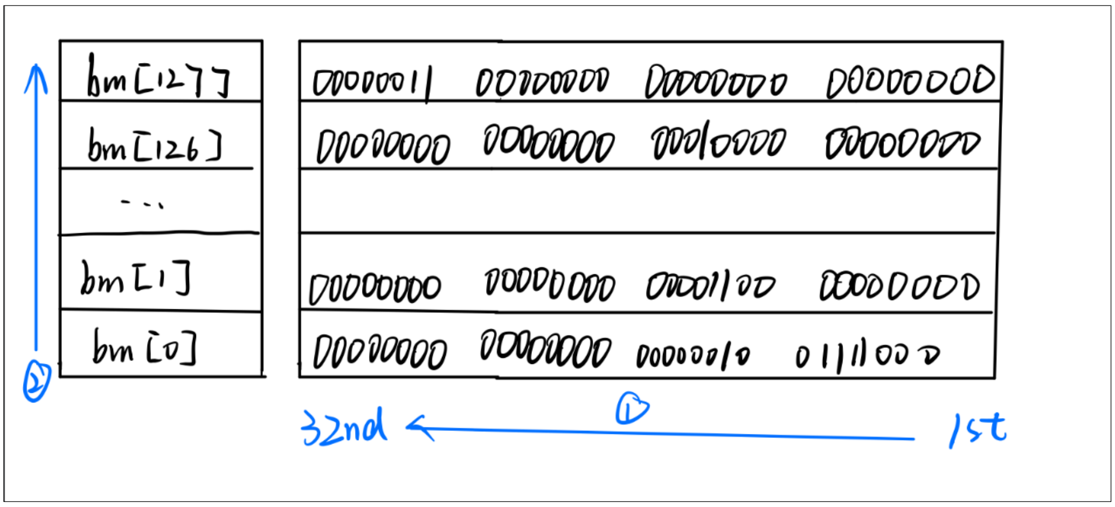


Figure5. FAT in memory

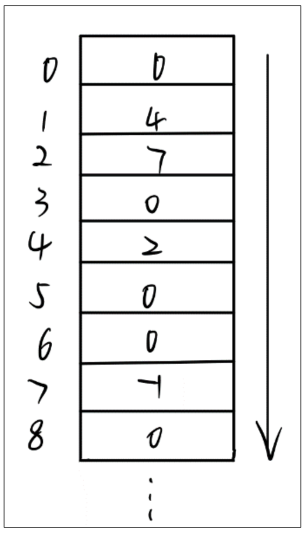


Figure6. File block table in memory

About how to manipulate bits in bitmap ("bit n" in bitmap represents n-th block in disk)

1. BM_ISCLR: (!(bm[(n)/BMEBITS] & (1 << ((n) % BMEBITS))))
   a) give a manipulate object "bit n" (0 < n < 4095)
   b) n / BMEBITS to find the row where "bit n" is in
   c) 1 << ((n) % BEMBITS) to create a 32-bit mask which only 1 in the position of the n-th bit
   d) Use logic and calculation to verify if the n-th bit is free or not

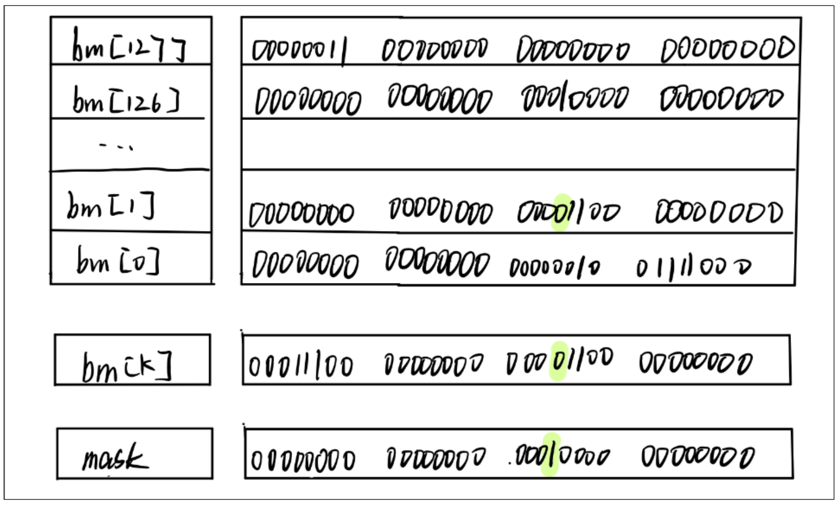


Figure7. BM_ISCLR

2. BM_SET: (!(bm[(n)/BMEBITS] & (1 << ((n) % BMEBITS))))
   a) give a manipulate object "bit n" (0 < n < 4095)
   b) n / BMEBITS to find the row where “bit n” is in
   c) 1 << ((n) % BEMBITS) to create a 32-bit mask which only 1 in the position of the n-th bit
   d) Set the n-th bit to one and maintain the rest bits by logic or

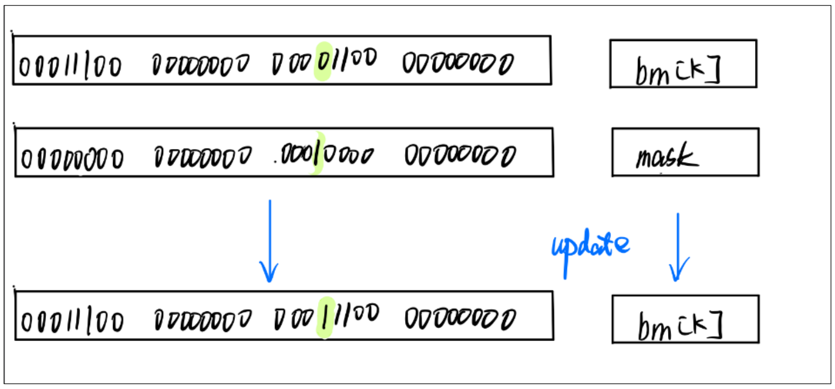


Figure8. BM_SET

3, BM_CLR: bm[(n)/BMEBITS] &= ~(1 << ((n) % BMEBITS)
   a) give a manipulate object "bit n" (0 < n < 4095)
   b) n / BMEBITS to find the row where “bit n” is in
   c) 1 << ((n) % BEMBITS) to create a 32-bit mask which only 1 in the position of the n-th bit
   d) Set “bit n” to zero and maintain the rest bits by logic and

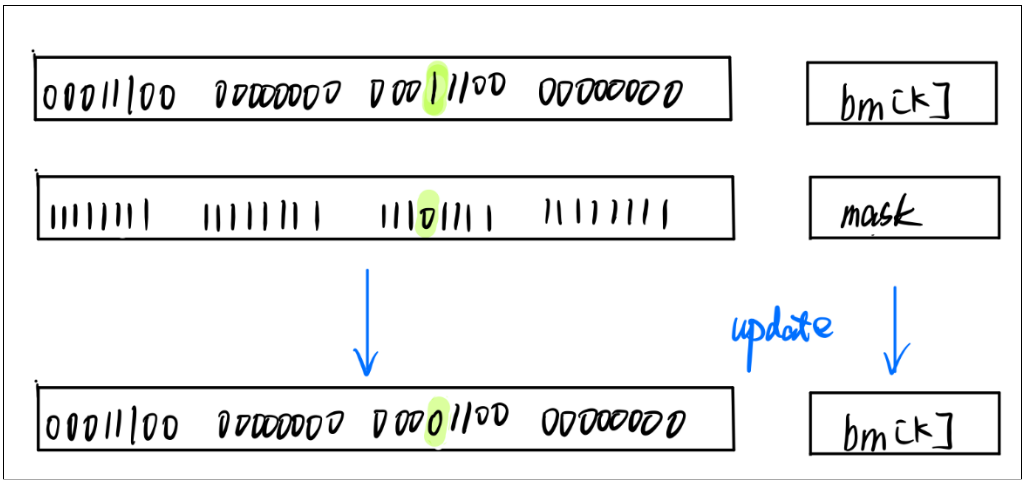


Figure9. BM_CLR

Free

- Free n consecutive blocks back to free space in bitmap. Stop when the n-th block is freed.
- Free n blocks in fbt, and stop when "-1" was detected.
- update # of free blocks

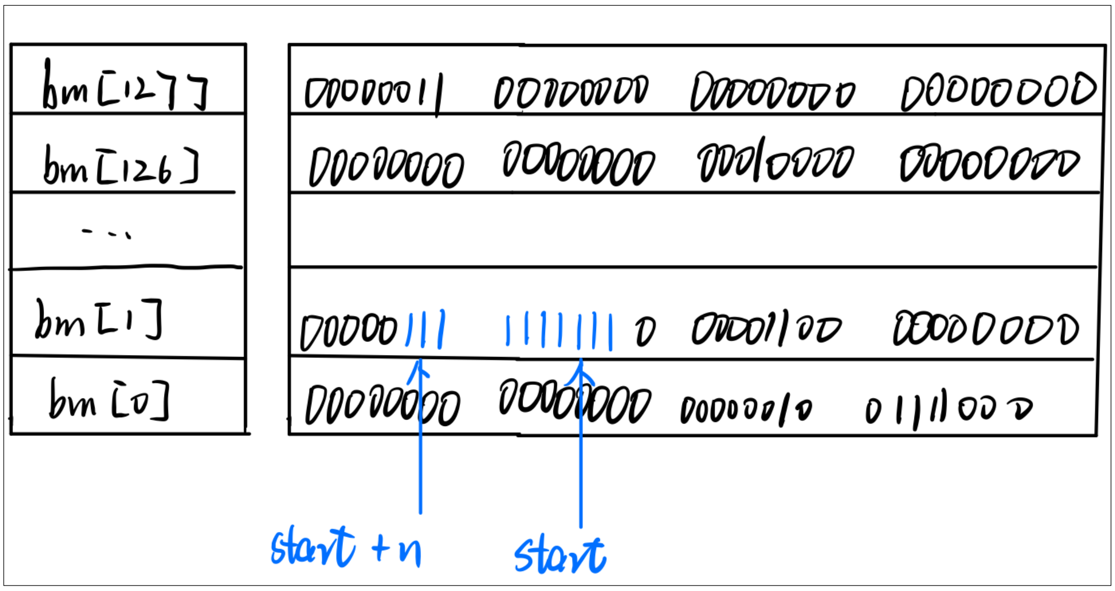


Figure10. Free n consecutive blocks with bitmap

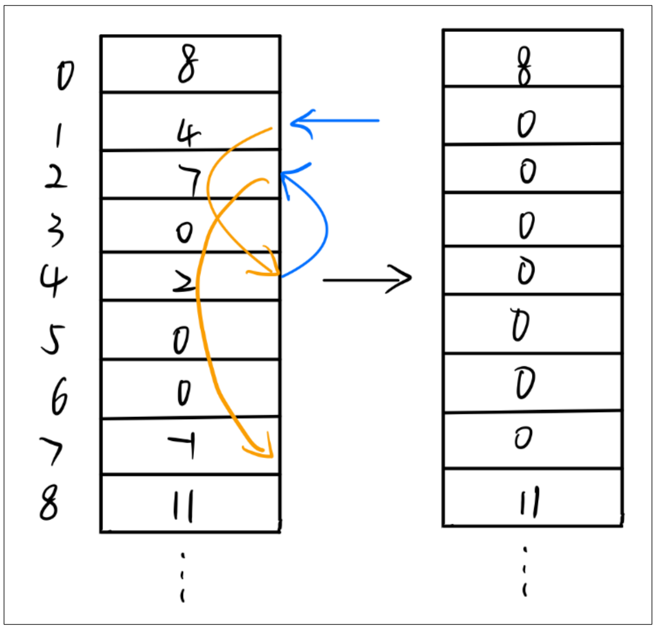


Figure11. Free a file of n blocks with fbt

Information request

- Dump: print the total information of bitmap or fbt
- fbt_list: Displays (prints) a list of blocks beginning with blkno. If blknois -1, displays a free block list.

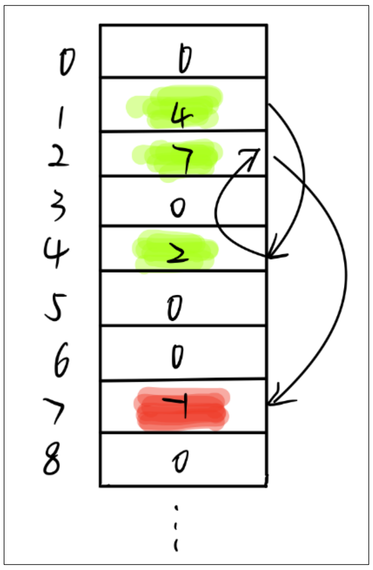


Figure12. Give requested information by retriving fbt

Verify

check if the real # of free blocks is consistent with the record (*bmfree) given by program

Test

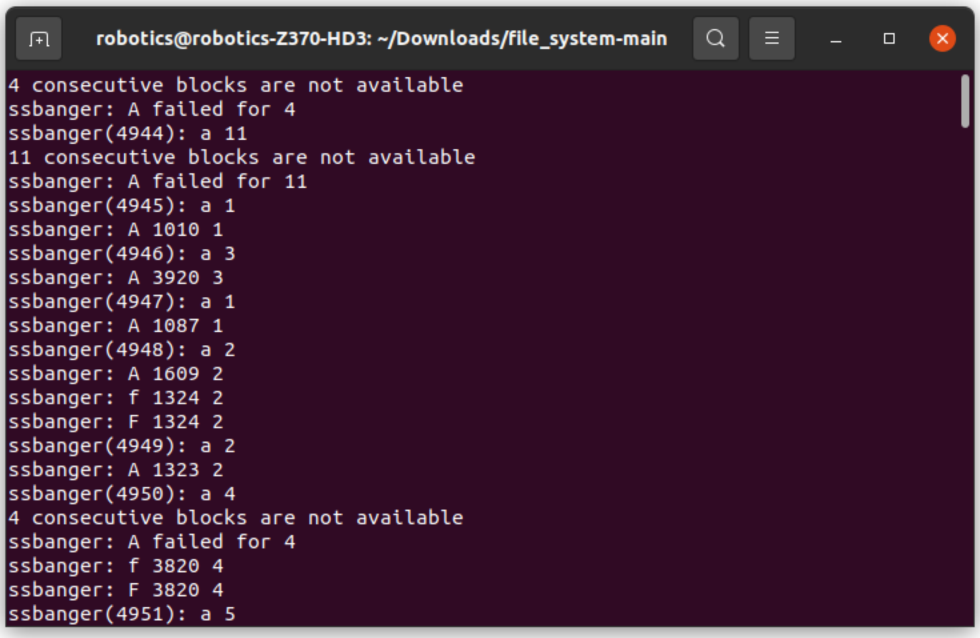
test on bitmap the results are attached with no problems
test on fbt the manual test seems no errors. But with banger(), according to the error log, it seem there are some problems caused an access beyond the limit of the array. I am sorry I had not found out the reason.

Note:

Output

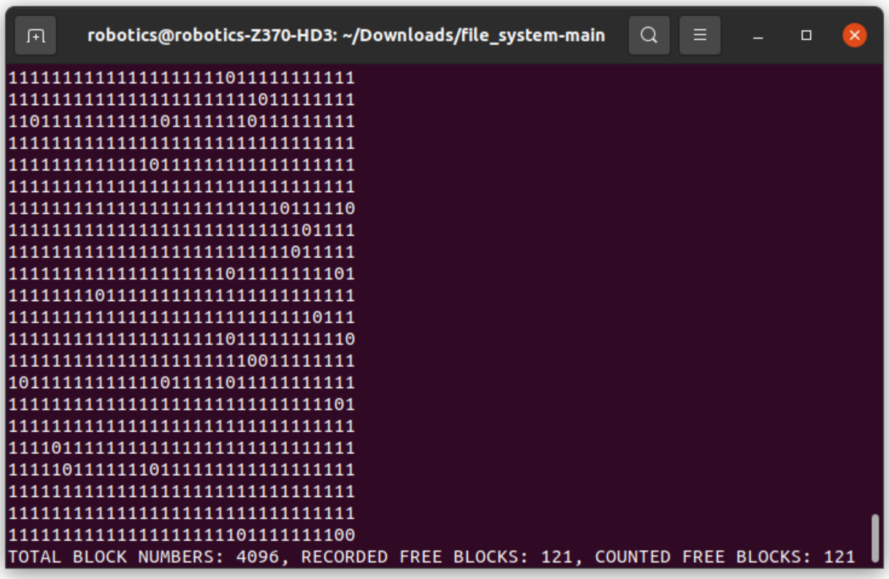
In this part, a simple description is provided to show what happened by run this file system.

1.The aim of including program for test is to generate some new files or free some existing files(with random size). This process will continue until there are no more space in virtual memory of this file system. In this stage, the duty of file system is to arrange these given files properly and make sure the memory is utilized efficiently.



```
4 consecutive blocks are not available
ssbanger: A failed for 4
ssbanger(4944): a 11
11 consecutive blocks are not available
ssbanger: A failed for 11
ssbanger(4945): a 1
ssbanger: A 1010 1
ssbanger(4946): a 3
ssbanger: A 3920 3
ssbanger(4947): a 1
ssbanger: A 1087 1
ssbanger(4948): a 2
ssbanger: A 1609 2
ssbanger: f 1324 2
ssbanger: F 1324 2
ssbanger(4949): a 2
ssbanger: A 1323 2
ssbanger(4950): a 4
4 consecutive blocks are not available
ssbanger: A failed for 4
ssbanger: f 3820 4
ssbanger: F 3820 4
ssbanger(4951): a 5
```

2. At last, the bitmap with detailed statistics will be shown. 1 for used memory block, and 0 for free memory block.



```
11111111111111111111011111111111
11111111111111111111111011111111
11011111111111011111110111111111
11111111111111111111111111111111
11111111111110111111111111111111
11111111111111111111111111111111
11111111111111111111111110111110
11111111111111111111111111101111
11111111111111111111111111011111
11111111111111111111011111111101
11111111011111111111111111111111
11111111111111111111111111110111
11111111111111111111011111111110
11111111111111111111110011111111
10111111111111011111011111111111
11111111111111111111111111111101
11111111111111111111111111111111
11110111111111111111111111111111
11111011111110111111111111111111
11111111111111111111111111111111
11111111111111111111111111111111
11111111111111111111101111111100
TOTAL BLOCK NUMBERS: 4096, RECORDED FREE BLOCKS: 121, COUNTED FREE BLOCKS: 121
```

Summary

By using technique like bit operation, I successfully make a simulation of file system with bitmap method. However the function is still inadequate for being a real file system. After making further investigation, I would like to enhance the functions to make a better simulation of file system.

References

[1] Andrew S. Tanenbaum. “Modern Operating Systems”, 4th ed., global ed., pp.282-284

[2] Abraham Silberschatz. “Operating System Concepts”, 10th ed., pp.573-582

[3] BMP file format - Wikipedia

[4] Bitmap Class (System.Drawing) | Microsoft Docs

[5] Free space management in Operating System - GeeksforGeeks