CO2017 — Surgery 6, File systems 2017–18

2018–02–19; R1156

Questions

Q1—File and directory representation

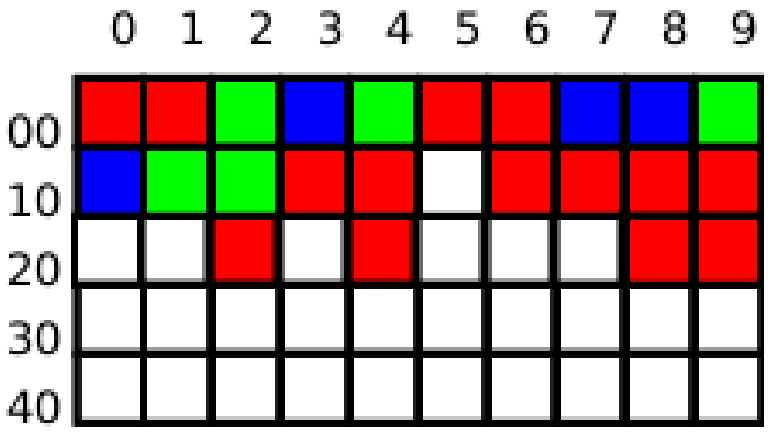

Figure 1: Disk layout of a small disk containing 3 files

Consider the diagram of a disk layout in Figure 1 which shows the blocks occupied by three files on a disk drive. The three files are called `red`, `blue` and `green`.

You can assume that the blocks are ordered in the files as shown. In other worde file `blue` is made from blocks 03, 07, 08, 10 in that order, *etc*.

The block size is 512 bytes, and 4 bytes are need to store one block address.

File Descriptors

For each allocation strategy, what would the file descriptor look like?

1. Contiguous allocation
2. Linked list
3. FAT
4. inode

File contents

For each allocation strategy, how would the block usage be recorded?

1. Linked list
2. FAT
3. inode

Model Answers to CO2017 Surgery 6

Q1

File Descriptors

1. Contiguous allocation — trick question, since the files are not laid out contiguously, this is meaningless
2. Linked list and FAT would be very similar:

   red block 00

   blue block 03

   green block 02
3. FAT (see above)
4. inode – file descriptors are just inode numbers. There is no simple way to predict what these will be in advance; they are assigned by the OS. In this case they could be:

   red inode 100

   blue inode 110

   green inode 105

File contents

1. Linked list

   Each block in each file would have space reserved for the block address of the next block. So

red

Block	points to block
00	01
01	05
05	06
etc.	
29	null

blue

Block	points to block
03	07
07	08
08	10
10	null

green

Block	points to block
02	04
04	09
09	11
11	12
12	null

2. FAT

   Maintain a separate data structure that contains all the links between blocks. The FAT is used for all the files at the same time.

   The FAT would be something like this:

From	to
00	01
01	05
02	04
03	07
04	09
05	06
06	13
07	08
08	10
09	11
10	null
11	12
12	null
13	14
14	16
etc.	
28	29
29	null

3. inode

   Suppose that the three files are using inodes 100, 110 and 105 respectively as described above.

   Then the inodes themselves would have something like this inside them:

   inode 100 for file red

name	red
size	14
etc.	other attribs
D1	00
D2	01
D3	05
D4	06
D5	13
D6	14
D7	16
D8	17
D9	18
D10	19
D11	22
D12	24
ID1	1000
ID2	null
ID3	null

Notice that file `red` has used up all 12 internal block addresses (D1 to D12) and needs 2 extra block addresses. For this, allocate block 1000 (for example; again the OS will actually make this allocation) and use "indirect block field 1" (`DS1`) in the inode to store its address. The whole of block 1000 will need to be allocated, even though it only needs the first few entries (a 512 byte block can hold 128 4-byte addresses); it will then contain:

28
29
null
null
etc. for 126 more entries

inode 110 for file blue

name	blue
size	4
etc.	other attribs
D1	03
D2	07
D3	08
D4	10
D5	null
etc.	for other direct blocks
ID1	null
ID2	null
ID3	null

Only 4 block addresses needed, so they can all be stored in the internal block addresses D1 to D4.

inode 105 for file green.

name	green
size	5
etc.	other attribs
D1	02
D2	04
D3	09
D4	11
D5	12
D6	null
etc.	for other direct blocks
ID1	null
ID2	null
ID3	null

Only 5 block addresses needed, so they can all be stored in the internal block addresses D1 to D5.