Names:

Rachel Peters - 816003509

Shivana Sookdeo - 816003030

Teneil Rodriguez - 816003952

Akil Hosang - 816003000

David Orr – 816003099

Course Code & Name: COMP 2604 – Operating Systems

Assignment 1

Year 2

Semester 2

* 1. The condition where the free list uses less space than the bitmap is when B is more than D multiplied by F (B > DF) or when F divided by B is less than 1 divided by D, where F divided by B is the fraction of free blocks (1/D > F/B).
     1. The free list can be recovered by using a recovery algorithm which will make a list of all the blocks in all the files and take the complement as the new free list.
     2. In a file allocation table file system, the problem cannot occur because there is no free list. The file allocation table file system uses scanning for free entries as it’s method of file recovery.
  2. A journaling file system keeps a log of what the file system is going to do before it does so that if the system crashes before it can do its planned work, upon rebooting the system can look in the log to see what was going on at the time of the crash and finish the job.
  3. Contiguous Allocation would not be suitable for this file since the file continuously grows and shrinks in size(dynamic file size), but using the contiguous allocation approach will prevent the files ability to grow in size.

Assuming the file sizes fluctuates often, the linked list allocation would not be suitable for this file because the time taken to read the file would be very slow. If the file rarely changes, this time may be acceptable in an idle processor.

Using the File Allocation Table would be suitable given that the disk size is relatively small and would make random access to it much easier. If the disk size is large then this method may not be suitable.

* 1. write-through cache is more suitable with a USB hard drive since at any moment, the USB hard drive can be disconnected from the computer at any moment which may leave the data on the disk inconsistent with the data that was updated in cache.
  2. hit rate = h%

time required to process a hit = 1 msec

probability of a miss= 1-h

time required to process a miss: 75 msec

Average Access Time = 1 msec \* h + (1 - h) \* 75 msec

= 1h + 75 -75h

= 75 - 74h

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| P2 | P5 | P4 | P3 | P1 |

0 3 7 13 22 37

AWT = (3+7+13+22+37)/5 = 82/5 = 16.4

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| P3 | P2 | P5 | P4 | P1 | P3 | P4 | P1 | P1 |

0 5 8 12 17 22 26 27 32 37

AWT = (8+12+26+27+37)/5 = 110/5 = 22

* + 1. Page 3 will be replaced
    2. Page 1 will be replaced
    3. Page 1 and 2 will be replaced

3. (a) **Contiguous allocation** stores a file in consecutive memory blocks. It can only store a file if the required number of blocks required to store the file can be allocated in a sequential manner and for a file to be added/created it is necessary to know its final size. Also, when files are removed, it creates holes or spaces throughout the disk. If the spaces are to be reused, it will be difficult to add the notes.txt file to the disk since the file size is varying from 4kb to 4m. Since the size of the file varies, so will the number of block require to store the file, this method of allocation may not be appropriate as allocating the number of blocks consecutively may not be possible.

**Linked-list allocation** does not require the block to be allocated consecutively and is able to store the file by linking any available blocks with the use of pointers to allocate the number of blocks needed to store the file. However, since the pointer takes up a few bytes, the amount of data storage in a block is no longer a power of 2. Therefore, it can be inefficient to store notes.txt if the file can only be written to blocks of the power of two.

**Linked-list allocation** that uses a memory table called the file allocation table (FAT), the links (block pointers) in the list are represented in the table which is indexed by block number this help to eliminate the disadvantage of linked-list allocation. Given a block number you could find the location to the next block with access to the FAT, and so on to find the other locations, a special value is used to notify the last block in the list. The FAT also contains a bit that identifies if a block is free or not. The disadvantage of this method is that the entire table must be in memory all the time to make it work.

(b) Number of cylinders: 200, numbered 0 to 199

Initial position of head: 30

Queue: 109, 190, 48, 123, 21, 135, 75, 77, 29

1. **FCFS (First-Come, First-Served)**

Order: 109, 190, 48, 123, 21, 135, 75, 77, 29

Total distance

= (109 – 30) + (190 – 109) + (190 – 48) + (123 – 48) + (123 – 21) + (135 – 21) + (135 – 75) + (77 – 75) + (77 – 29)

= 703 cylinders

1. **SSTF (Shortest-Seek-Time-First)**

Order: 29, 21, 48, 75, 77, 109, 123, 135, 190

Total distance

= (30 – 21) + (29 – 21) + (48 – 29) + (75 – 48) + (77 – 75) + (109 – 77) + (123 – 109) + (135 – 123) + (190 – 135)

= 178 cylinders

1. **C-SCAN (Circular Scan)**

Order: 48, 75, 77, 109, 123, 135, 190, 199, 21, 29

Total distance

= (48 – 30) + (74 – 48) + (77 – 75) + (109 – 77) + (123 – 109) + (135 – 123) + (190 – 135) + (199 – 190) + (199 – 0) + (21 – 0) + (29 – 21)

= 397 cylinders

1. **C-LOOK (Circular Look)**

Order: 48, 75, 77, 109, 123, 135, 190, 199, 21, 29

Total distance

= (48 – 30) + (74 – 48) + (77 – 75) + (109 – 77) + (123 – 109) + (135 – 123) + (190 – 135) + (190 – 21) + (29 – 21)

= 337 cylinders

4) a)

FAT Table entries:

(14, 18); (15, 17); (16, 23); (17, 21); (18, 20); (19, 15); (20, -1); (21, -1); (22, 19); (23, 14)

The first data block of f1 = 22 and f2 = 16

The data blocks allotted to the two files on disk are:

f1: 22, 19, 15, 17, 21

f2: 16, 23, 14, 18, 20

b)

Since the i-node for the root is in the directory already, 10 disk operations are needed which are:

1. Fetch / directory
2. Fetch i-node for usr
3. Fetch /usr directory
4. Fetch i-node for ast
5. Fetch /usr/ast
6. Fetch i-node for courses
7. Fetch /usr/ast/courses
8. Fetch i-node for os
9. Fetch /usr/ast/courses/os
10. Fetch i-node for handout.txt

c) i)

The parameters of the creat() function are a name and a protection mode

The name string defines the name of the file to be created

The protection mode is an integer that specifies the file permissions that will be used to create the file.

The return value is an integer representing the file descriptor for the file created.

ii)

int fd = creat(“data.txt”, 0760)