Daniel Peterson 109091561

4.Is the open system call in UNIX absolutely essential? What would the consequences be of not having it?

**The open system call opens the file and allows the system to fetch the attributes and list of disk addresses into main memory for faster access in future calls. If the open system call did not exist, then the name of the file must be specified every time a read or seek operation is called. This value would have to be looked up each time the read or seek is called, which would add so much unnecessary overhead time. Therefore, the call is not absolutely essential, but it makes reading files so much faster.**

5.Systems that support sequential files always have an operation to rewind files. Do systems that support random-access files need this, too?

**Systems that support random access files do not need the operation to rewind files. Sequential files need a rewind operation in case the file needed to be read more than once. In random access files, the file can be read as much as is needed, so no operation is necessary.**

9.In UNIX and Windows, random access is done by having a special system call that moves the ‘‘current position’’ pointer associated with a file to a given byte in the file. Propose an alternative way to do random access without having this system call.

**An alternative way to do this random access is by using a *read* operation to specify where to start reading a file. This operation gives the position in the file to start reading at as opposed to the seek operation which moves the pointer to the place in the file to be read.**

10.Consider the directory tree of Fig. 4-8. If */usr/jim* is the working directory, what is the absolute path name for the file whose relative path name is *../ast/x*?

**usr/ast/x because the parent (..) of /ast/ is /usr/ since the working directory is /usr/jim, the .. operation goes to the parent, which is /usr/.**

12.Describe the effects of a corrupted data block for a given file for: (a) contiguous, (b) linked, and (c) indexed (or table based).

**Corrupted data blocks in contiguous systems may lead to issues if there are no free holes to copy the data blocks into since the blocks for a particular file are intended to be stored on after the other. For linked lists, if a block is corrupted, then the reference to the next block in the file may be unavailable, which will prevent reading the rest of the file. In table based, if the i-node is not corrupted, then any of the blocks of the file can be found. If the i-node is corrupted, then the file will be unable to be opened and the i-node will not be in memory.**

13**.** One way to use contiguous allocation of the disk and not suffer from holes is to compact the disk every time a file is removed. Since all files are contiguous, copying a file requires a seek and rotational delay to read the file, followed by the transfer at full speed. Writing the file back requires the same work. Assuming a seek time of 5 msec, a rotational delay of 4 msec, a transfer rate of 80 MB/sec, and an average file size of 8 KB, how long does it take to read a file into main memory and then write it back to the disk at a new location? Using these numbers, how long would it take to compact half of a 16-GB disk?

**The transfer takes 5msec + 4msec to start. 8KB = 2^13, 8MB/sec = 2^23 / sec**

**2^13 / 2^23 /sec = 2^-10 sec = 0.977msec to read and same to write**

**9.977 + 9.977 = 19.954 msec to do a complete copy**

**16GB disk = 2^34, half is 8GB or 2^33**

**2^33 (space)/ 2^13(file size) = 2^20(files)**

**2^20 \* 19.954 \* 10^-3 sec = 20,923 seconds = 348.7 minutes = 5.81 hours**

**Almost 6 hours to compact half of a disk each time that a file is removed.**

16**.** Consider the i-node shown in Fig. 4-13. If it contains 10 direct addresses and these were 8 bytes each and all disk blocks were 1 KB, what would the largest possible file be?

**1KB/8 = 128 addresses**

**10 Direct + 128 = 138 addresses**

**138 \* 1KB = 138KB for largest possible file**

17.For a given class, the student records are stored in a file. The records are randomly accessed and updated. Assume that each student’s record is of fixed size. Which of the three allocation schemes (contiguous, linked and table/indexed) will be most appropriate?

**Linked list has extremely slow random access so it would not be a good choice. Since the student’s record is a fixed size, contiguous allocation would be most appropriate since it has extremely fast access time relative to tables.**

18.Consider a file whose size varies between 4 KB and 4 MB during its lifetime. Which

of the three allocation schemes (contiguous, linked and table/indexed) will be most appropriate?

**Contiguous would not be appropriate because a hole size which could fit the maximum file size would be needed in order to store the file. The best allocation scheme is likely linked list allocation due to the fact that it can have varying size files and the files can easily be read sequentially.**

20**.** Two computer science students, Carolyn and Elinor, are having a discussion about i-nodes. Carolyn maintains that memories have gotten so large and so cheap that when a file is opened, it is simpler and faster just to fetch a new copy of the i-node into the i-node table, rather than search the entire table to see if it is already there. Elinor disagrees. Who is right?

**If what Carolyn says is true, then the act of fetching a new copy of the i-node must be faster than searching the i-node table to see if it exists. This means that she believes making a disk lookup is faster than looking up the table in cache, which is incorrect. Therefore, Elinor is correct.**

22**.** Explain how hard links and soft links differ with respective to i-node allocations.

**Hard links are links where the original i-node is assigned to the link. The original file and the hard link will have the same i-node number. This means that if the original file is deleted, the link still has the i-node so the data can still be accessed. Soft links are links where a new i-node is assigned to the link. The original file and the soft link have different i-node numbers. The link’s i-node points to the original i-node. This means that if the original file is deleted, then the link only points to the original i-node, which is now gone, so the data cannot be read.**

25**.** The beginning of a free-space bitmap looks like this after the disk partition is first formatted: 1000 0000 0000 0000 (the first block is used by the root directory). The system always searches for free blocks starting at the lowest numbered block, so after writing file *A*, which uses six blocks, the bitmap looks like this: 1111 1110 0000 0000. Show the bitmap after each of the following additional actions:

(a) File *B* is written, using five blocks.

(b) File *A* is deleted.

(c) File *C* is written, using eight blocks.

(d) File *B* is deleted.

**a)1111 1110 0000 0000 ->add 5 1s ->1111 1111 1111 0000**

**b)1111 1111 1111 0000 ->remove 6 1s added from A ->1000 0001 1111 0000**

**c) 1000 0001 1111 0000 ->add 8 1s, 5 for where A was and 3 extra ->1111 1111 1111 1100**

**d)1111 1111 1111 1100 ->remove 6 1s added from B ->1111 1110 0000 1100**

29.Suppose that file 21 in Fig. 4-25 was not modified since the last dump. In what way would the four bitmaps of Fig. 4-26 be different?

**In the first phase, since file 21 is not a directory and is not modified, it would not be colored in. In the second phase, and beyond, 21 would remain not colored in since it is still not modified. No other files or directories would be affected by this change.**

30.It has been suggested that the first part of each UNIX file be kept in the same disk block as its i-node. What good would this do?

**The benefit of keeping the first part of each UNIX file on the same disk block as its i-node is that the disk would need to be accessed less in order to read a file. One less disk access is required no matter what the file size is.**

**35.** Consider a disk that has 10 data blocks starting from block 14 through 23. Let there be 2 files on the disk: f1 and f2. The directory structure lists that the first data blocks of f1 and f2 are respectively 22 and 16. Given the FAT table entries as below, what are the data blocks allotted to f1 and f2? (14,18); (15,17); (16,23); (17,21); (18,20); (19,15); (20, -1); (21, -1); (22,19); (23,14). In the above notation, (*x*, *y*) indicates that the value stored in table entry *x* points to data block *y*.

**F1:**

**22 -> 19 -> 15 ->17 -> 21 -> EOF**

**F2:**

**16 -> 23 -> 14 -> 18 -> 20 -> EOF**

36**.** Consider the idea behind Fig. 4-21, but now for a disk with a mean seek time of 6 msec, a rotational rate of 15,000 rpm, and 1,048,576 bytes per track. What are the data rates for block sizes of 1 KB, 2 KB, and 4 KB, respectively?

**15,000 rpm is 4msec**

**6msec + 2msec + (1KB/1,048,576) \* 4msec = 8.00390625**

**6msec + 2msec + (2KB/1,048,576) \* 4msec = 8.0078125**

**6msec + 2msec + (4KB/1,048,576) \* 4msec = 8.015625**

**1KB/8.00390625 = 127,937 KB/sec**

**2KB/8.0078125 = 255,750 KB/sec**

**4KB/8.015625 = 511,002 KB/sec**

40.A UNIX file system has 4-KB blocks and 4-byte disk addresses. What is the maximum file size if i-nodes contain 10 direct entries, and one single, double, and triple indirect entry each?

**Direct: 10\*4KB = 40,960**

**Single: (4KB/4) \* 4KB = 4,194,304**

**Double: (4KB/4) \* (4KB/4) \* 4KB = 4,294,967,296**

**Triple: (4KB/4) \* (4KB/4) \* (4KB/4) \* 4KB = 4,398,046,511,104**

**40,960 + 4,194,304 + 4,294,967,296 + 4,398,046,511,104 = 4,402,345,713,664**

41.How many disk operations are needed to fetch the i-node for a file with the path name */usr/ast/courses/os/handout.t*? Assume that the i-node for the root directory is in memory, but nothing else along the path is in memory. Also assume that all directories fit in one disk block.

**Ten disk operations are needed to fetch the i-node for the file. One fetch is required for each directory in the path (root, usr, ast, courses, os) so there are 5 total for the directories. Then one fetch is required for each of the i-nodes which point to the next location in the path (usr from root, ast from usr, courses from ast, os from courses, and handout.t from os) so there are 5 more fetches. This is 10 total fetches.**

42**.** In many UNIX systems, the i-nodes are kept at the start of the disk. An alternative de- sign is to allocate an i-node when a file is created and put the i-node at the start of the first block of the file. Discuss the pros and cons of this alternative.

**If the first block of the file contains the i-node then when the file is being read, the disk would not need to be read as much because the i-node is already present and can be read along with the data in one operation. A downside to this is that the entire block must be read for each i-node which slows reading down. Additionally, if a file is designed to fit a particular block size, it may no longer fit if the i-node is inserted.**