**PROBLEMS**

**CHAPTER 4. FILE SYSTEMS**

1. 1. Give five different path names for the file */etc/passwd*. (*Hint*: Think about the direc-tory entries ‘‘.’’ and ‘‘..’’.)
2. 2. In Windows, when a user double clicks on a file listed by Windows Explorer, a pro-gram is run and given that file as a parameter. List two different ways the operating system could know which program to run.
3. 3. In early UNIX systems, executable files (*a.out* files) began with a very specific magic number, not one chosen at random. These files began with a header, followed by the text and data segments. Why do you think a very specific number was chosen for ex-ecutable files, whereas other file types had a more-or-less random magic number as the first word?
4. 4. Is the open system call in UNIX absolutely essential? What would the consequences be of not having it?
5. 5. Systems that support sequential files always have an operation to rewind files. Do sys-tems that support random-access files need this, too?
6. 6. Some operating systems provide a system call rename to give a file a new name. Is there any difference at all between using this call to rename a file and just copying the file to a new file with the new name, followed by deleting the old one?
7. 7. In some systems it is possible to map part of a file into memory. What restrictions must such systems impose? How is this partial mapping implemented?
8. 8. A simple operating system supports only a single directory but allows it to have arbi-trarily many files with arbitrarily long file names. Can something approximating a hier-archical file system be simulated? How?
9. 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.
10. 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*?
11. 11. Contiguous allocation of files leads to disk fragmentation, as mentioned in the text, be-cause some space in the last disk block will be wasted in files whose length is not an integral number of blocks. Is this internal fragmentation or external fragmentation? Make an analogy with something discussed in the previous chapter.
12. 12. Describe the effects of a corrupted data block for a given file for: (a) contiguous, (b) linked, and (c) indexed (or table based).
13. 13. One way to use contiguous allocation of the disk and not suffer from holes is to com-pact 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?
14. 14. In light of the answer to the previous question, does compacting the disk ever make any sense?
15. 15. Some digital consumer devices need to store data, for example as files. Name a modern device that requires file storage and for which contiguous allocation would be a fine idea.
16. 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 1024 KB, what would the largest possible file be?
17. 17. For a given class, the student records are stored in a file. The records are randomly ac-cessed 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 ap-propriate?
18. 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 ap-propriate?
19. 19. It has been suggested that efficiency could be improved and disk space saved by stor-ing the data of a short file within the i-node. For the i-node of Fig. 4-13, how many bytes of data could be stored inside the i-node?
20. 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 dis-agrees. Who is right?
21. 21. Name one advantage of hard links over symbolic links and one advantage of symbolic links over hard links.
22. 22. Explain how hard links and soft links differ with respective to i-node allocations.
23. 23. Consider a 4-TB disk that uses 4-KB blocks and the free-list method. How many block addresses can be stored in one block?
24. 24. Free disk space can be kept track of using a free list or a bitmap. Disk addresses re-quire *D* bits. For a disk with *B* blocks, *F* of which are free, state the condition under which the free list uses less space than the bitmap. For *D* having the value 16 bits, express your answer as a percentage of the disk space that must be free.
25. 25. The beginning of a free-space bitmap looks like this after the disk partition is first for-matted: 1000 0000 0000 0000 (the first block is used by the root directory). The sys-tem 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:
    1. (a) File *B* is written, using five blocks.
    2. (b) File *A* is deleted.
    3. (c) File *C* is written, using eight blocks.
    4. (d) File *B* is deleted.
26. 26. What would happen if the bitmap or free list containing the information about free disk blocks was completely lost due to a crash? Is there any way to recover from this disas-ter, or is it bye-bye disk? Discuss your answers for UNIX and the FAT-16 file system separately.
27. 27. Oliver Owl’s night job at the university computing center is to change the tapes used for overnight data backups. While waiting for each tape to complete, he works on writ-ing his thesis that proves Shakespeare’s plays were written by extraterrestrial visitors. His text processor runs on the system being backed up since that is the only one they have. Is there a problem with this arrangement?
28. 28. We discussed making incremental dumps in some detail in the text. In Windows it is easy to tell when to dump a file because every file has an archive bit. This bit is miss-ing in UNIX. How do UNIX backup programs know which files to dump?
29. 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?
30. 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?
31. 31. Consider Fig. 4-27. Is it possible that for some particular block number the counters in *both* lists have the value 2? How should this problem be corrected?
32. 32. The performance of a file system depends upon the cache hit rate (fraction of blocks found in the cache). If it takes 1 msec to satisfy a request from the cache, but 40 msec to satisfy a request if a disk read is needed, give a formula for the mean time required to satisfy a request if the hit rate is *h*. Plot this function for values of *h* varying from 0 to 1.0.
33. 33.For an external USB hard drive attached to a computer, which is more suitable: a write-through cache or a block cache?
34. 34. Consider an application where students’ records are stored in a file. The application takes a student ID as input and subsequently reads, updates, and writes the correspond-ing student record; this is repeated till the application quits. Would the "block read-ahead" technique be useful here?
35. 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*.

1. 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?
2. 37. A certain file system uses 4-KB disk blocks. The median file size is 1 KB. If all files were exactly 1 KB, what fraction of the disk space would be wasted? Do you think the wastage for a real file system will be higher than this number or lower than it? Explain your answer.
3. 38. Given a disk-block size of 4 KB and block-pointer address value of 4 bytes, what is the largest file size (in bytes) that can be accessed using 10 direct addresses and one indi-rect block?
4. 39. Files in MS-DOS have to compete for space in the FAT-16 table in memory. If one file uses *k* entries, that is *k* entries that are not available to any other file, what constraint does this place on the total length of all files combined?
5. 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?
6. 41. How many disk operations are needed to fetch the i-node for afile with the path name */usr/ast/courses/os/handout.t*? Assume that the i-node for the root directory is in mem-ory, but nothing else along the path is in memory. Also assume that all directories fit in one disk block.
7. 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.
8. 43. Write a program that reverses the bytes of a file, so that the last byte is now first and the first byte is now last. It must work with an arbitrarily long file, but try to make it reasonably efficient.
9. 44. Write a program that starts at a given directory and descends the file tree from that point recording the sizes of all the files it finds. When it is all done, it should print a histogram of the file sizes using a bin width specified as a parameter (e.g., with 1024, file sizes of 0 to 1023 go in one bin, 1024 to 2047 go in the next bin, etc.).
10. 45. Write a program that scans all directories in a UNIX file system and finds and locates all i-nodes with a hard link count of two or more. For each such file, it lists together all file names that point to the file.
11. 46. Write a new version of the UNIX *ls* program. This version takes as an argument one or more directory names and for each directory lists all the files in that directory, one line per file. Each field should be formatted in a reasonable way given its type. List only the first disk address, if any.
12. 47. Implement a program to measure the impact of application-level buffer sizes on read time. This involves writing to and reading from a large file (say, 2 GB). Vary the appli-cation buffer size (say, from 64 bytes to 4 KB). Use timing measurement routines (such as *gettimeofday* and *getitimer* on UNIX) to measure the time taken for different buffer sizes. Analyze the results and report your findings: does buffer size make a difference to the overall write time and per-write time?
13. 48. Implement a simulated file system that will be fully contained in a single regular file stored on the disk. This disk file will contain directories, i-nodes, free-block infor-mation, file data blocks, etc. Choose appropriate algorithms for maintaining free-block information and for allocating data blocks (contiguous, indexed, linked). Your pro-gram will accept system commands from the user to create/delete directories, cre-ate/delete/open files, read/write from/to a selected file, and to list directory contents.