Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich
Spring Term 2014

Operating Systems and Networks
Assignment 8

Assigned on: **10th April 2014**
Due by: **17th April 2014**

1 File Systems

a) *Is the "open" system call in UNIX absolutely essential? What would be the consequences of not having it?*

Answer: If there were no "open" system call, it would be necessary to specify the name of the file to be accessed for every read operation. The system would then have to fetch the i-node for it, although it could be cached. One issue that quickly arises is when to flush the inode back to disk. It could be based on a timeout, however it would be clumsy. Overall, it may work, but with much more overhead involved.

b) *Some OSes 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?*

Answer: Yes, the "rename" call does not change the creation time, or the time of last modification, but creating a new file causes it to get the current time as both the creation and last modification times. Also, if the disk is full, the copy will fail.

c) *Name one advantage of hard links over symbolic links and one advantage of symbolic links over hard links.*

Answer: Hard links do not require any additional disk space, just a counter in the inode to keep track of how many there are. On the other hand, symbolic links need space to store the name of the file pointed to it. Symbolic links can point to files on other machines, even over the internet. Hard links are restricted to pointing to files within their own partition.

d) *It has been suggested that the first part of each file be kept in the same disk block as its inode. What good would this do?*

Answer: Often, files are short. If the entire file fit in the same block as the inode, only one disk access would be needed to read the file, instead of two, as is presently the case. Even for longer files there would be a gain, since one fewer disk accesses would be needed.

e) *An Operating System only supports a single directory, but allows that directory to have arbitrarily many files with arbitrarily long file names. Can something approximating a hierarchical file system be simulated? How?*

Answer: One way to simulate that is to append to each file name the name of directory that contains it.
For example: UsrStudentsZivaPrivateFileX

f) *Systems that support sequential files always have an operation to rewind files. Do systems that support random access files need this too?*

Answer: No, random access of files does not need the "rewind" operation since if you want to read the file again, you can just access byte 0.

g) *Contiguous allocation of files leads to disk fragmentation. Is this internal fragmentation or external fragmentation?*

Answer: Contiguous allocation of files leads to disk internal fragmentation, since disk are divided into blocks the only disk space waste is in the block itself.

h) *One way to use contiguous allocation of 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 4msec, a transfer rate of 8MB/sec and an average file size of 8KB, how long does it take to read a file into main memory then write it back to the disk at a new location? Using these numbers, how long would it take to compact half of a 16GB disk?*

Answer: It takes 9msec to start the transfer (due to 5msec seek and 4msec rotation delay). To read 2^{13} bytes (8KB) at the transfer rate of 2^{23} bytes/sec (8MB/sec) requires 2^{-10} sec (0.977msec). Hence the total time to seek, rotate and transfer is 9.977msec. Writing back takes another 9.977msec. Thus copying an average file takes 19.954msec.

To compact half of a 16GB disk would involve copying 8GB of storage, which is 2^{20} files. At 19.954 msec per file, this takes 20,923 seconds, which is 5.8 hours. Clearly, compacting the disk after every file removal is not a great idea.

i) *Consider the inode structure with 10 direct addresses and one indirect address. If the inode contains 10 direct addresses of 4 bytes each and all disk blocks are 1024 Bytes, what is the largest possible file? Consider 32-bit addressing.*

Answer: With the 10 direct addresses we get file size of 10KB. With a single indirect address we can point to one block of addresses which means $\frac{1024}{32/4} = 256$ addresses to blocks. This also means that we can address 256KB with the indirect addressing. All together, we can have the largest possible file of $256 + 10 = 266$KB.

j) *Free disk space can be kept track of using free list or a bit map. Disk addresses require 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 bit map. For D having the value 16 bits, express your answer as a percentage of the disk space that must be free.*

Answer: The bit map requites B bits. The free list requires DF bits. The free list requires fewer bits if $DF < B$. Alternatively, the free list is shorter if $\frac{F}{B} < \frac{1}{D}$, where $\frac{F}{B}$ is the fraction of blocks free. For 16-bit disk addresses the free list is shorter if 6 percent or less of the disk is free.

k) *The beginning of the 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 a file A which uses 6 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 5 blocks.*
(b) *File A is deleted.*
(c) *File C is written, using 8 blocks.*
(d) *File B is deleted.*

Answer:

(a) `1111 1111 1111 0000`
(b) `1000 0001 1111 0000`

(c) `1111 1111 1111 1100`

(d) `1111 1110 0000 1100`