CSCI 3323 (Principles of Operating Systems), Fall 2011

Homework 6

Credit: 10 points.

1 Reading

Be sure you have read Chapter 4.

2 Problems

Answer the following questions. You may write out your answers by hand or using a word processor or other program, but please submit hard copy, either in class or in my mailbox in the department office.

1. (5 points) The textbook describes more than one strategy for keeping track of free blocks in a file system (free blocks, bitmaps, and FATs). All of these strategies rely on information that is kept both on disk and in memory, sometimes with the most-current information only in memory. What would happen if the copy on disk of whatever data structure is used to keep track of free blocks was lost or damaged because of a system crash — is there a way to recover, or do you have to just reformat the disk and hope you backed up any really important files? Answer separately for MS-DOS FAT-16 (which uses a FAT) and UNIX V7 filesystems (which uses one of the other strategies).

2. (5 points) Consider a UNIX filesystem (as described in section 4.5.3) in which each i-node contains 10 direct entries, one single-indirect entry, one double-indirect entry, and one triple-indirect entry. If a block is 1KB (1024 bytes) and a disk addresses is 4 bytes, what is the maximum file size, in KB? (*Hint:* Use the blocksize and size of disk addresses to determine how many entries each indirect block contain.)

3 Programming Problems

For extra credit, do one or more of the following programming problems. You will end up with at least one code file per problem. Submit your program source (and any other needed files) by sending mail to `bmassing@cs.trinity.edu`, with each file as an attachment. Please use a subject line that mentions the course number and the assignment (e.g., "csci 3323 homework 6"). You can develop your programs on any system that provides the needed functionality, but I will test them on one of the department's Linux machines, so you should probably make sure they work in that environment before turning them in.

1. (Optional — up to 5 extra-credit points) Write a program that given a directory D, blocksize B, and maximum number of blocks M as command-line arguments prints out how many files in D and its subdirectories are of size B or less, how many are of size between B and $2B$, etc., up to size MB. (This might be useful in getting an idea of what size files are typical, so if you had a choice of blocksize you would know what choice might make the most sense.) Include directories and symbolic links (but count the size of the link and not the file/directory

it links to). Also turn in output of running this program on your home directory in `/users` with B and M as below.

Here is sample output for running the program with D = `/lib`, $B = 4096$, and $M = 20$, on Xena00:

```
Results for directory /lib with blocksize 4096:

               814 files of size          1 blocks
               902 files of size          2 blocks
               866 files of size          3 blocks
               725 files of size          4 blocks
               557 files of size          5 blocks
               424 files of size          6 blocks
               273 files of size          7 blocks
               225 files of size          8 blocks
               204 files of size          9 blocks
               184 files of size         10 blocks
               122 files of size         11 blocks
               140 files of size         12 blocks
               145 files of size         13 blocks
                80 files of size         14 blocks
                89 files of size         15 blocks
                64 files of size         16 blocks
                85 files of size         17 blocks
                76 files of size         18 blocks
                45 files of size         19 blocks
                39 files of size         20 blocks
               758 files of size         21 blocks or more
```

(Of course, you won't be able to examine files in directories you don't have access to. Just print error messages for files/directories you can't access.)

To get maximum points, your program should be in C or C++ and make no use of system commands such as `ls`. (You can use another language, or even write a shell script, but you will get fewer points.) Library functions `opendir`, `readdir`, and `lstat` will probably be helpful. You might also be interested in functions `chdir` and `strerror`. These functions are described by man pages. (Remember also that `man -a foo` gives all man pages for `foo`. This can be helpful if there is both a command `foo` and a functionfoo.)

Here is some starter code[1] that parses/checks the command-line arguments.

2. (Optional — up to 5 extra-credit points) Write a program that given a directory D as a command-line argument prints all the "broken" symbolic links in D or any of its subdirectories — that is, symbolic links that point to a file that doesn't exist. Here is sample output for running the program with D = `/users/bmassing/Local/HTML-Documents/CS4320/Homeworks/HW04/Problems`:

```
Broken symbolic links in /users/bmassing/Local/HTML-Documents/CS3323/Homeworks/HW06/Problems:

/users/bmassing/Local/HTML-Documents/Classes/CS3323_2011fall/Homeworks/HW06/Problems/TestData/foobar
/users/bmassing/Local/HTML-Documents/Classes/CS3323_2011fall/Homeworks/HW06/Problems/TestData/barfoo
```

[1] `http://www.cs.trinity.edu/~bmassing/Classes/CS3323_2011fall/Homeworks/HW06/Problems/filesizes.c`

(Again, you won't be able to examine files in directories you don't have access to, so just print error messages. You should be able to access everything in the above directory, however. If you want to create some test data of your own, remember that to make a symbolic link called `sym` pointing to `foo`, you type `ln -s foo sym`.)

To get maximum points, your program should be in C or C++ and make no use of system commands such as `ls`. (You can use another language, or even write a shell script, but you will get fewer points.) The library routines mentioned for the previous problem may be helpful. The starter code may also be helpful, in reminding you how to access command-line arguments in C.

3. (Optional — up to 5 extra-credit points) Write a program that given a directory D as a command-line argument finds all the files in D or any of its subdirectories to which there are two or more hard links and prints, for each of them, all the paths within D that point to that file. Here is sample output for running the program with D = `/users/bmassing/Local/HTML-Documents/CS4320/Homeworks/HW06/Problems`:

```
Files with multiple hard links in /users/bmassing/Local/HTML-Documents/CS3323/Homeworks/HW06/Problems:

/users/bmassing/Local/HTML-Documents/Classes/CS3323_2011fall/Homeworks/HW06/Problems/TestData/bbbb
/users/bmassing/Local/HTML-Documents/Classes/CS3323_2011fall/Homeworks/HW06/Problems/TestData/b
/users/bmassing/Local/HTML-Documents/Classes/CS3323_2011fall/Homeworks/HW06/Problems/TestData/bb
/users/bmassing/Local/HTML-Documents/Classes/CS3323_2011fall/Homeworks/HW06/Problems/TestData/bbb

/users/bmassing/Local/HTML-Documents/Classes/CS3323_2011fall/Homeworks/HW06/Problems/TestData/dd
/users/bmassing/Local/HTML-Documents/Classes/CS3323_2011fall/Homeworks/HW06/Problems/TestData/d
```

This output means that the two pathnames in the first group reference the same file, the four pathnames in the second group reference the same file, etc. Output can be in any order as long as paths that reference the same file are grouped together. (Again, you won't be able to examine files in directories you don't have access to, so just print error messages. You should be able to access everything in the above directory, however. If you want to create some test data of your own, remember that to make a hard link called `sym` pointing to `foo`, you type `ln foo sym`.)

To get maximum points, your program should be in C or C++ and make no use of system commands such as `ls`. (You can use another language, or even write a shell script, but you will get fewer points.) The library routines mentioned for the previous problems may be helpful. The starter code may also be helpful, in reminding you how to access command-line arguments in C.