EECS 211 – Spring Quarter, 2010
Program 8

Due Thursday, June 10th, 2010 at 11:59PM

Background:

Recall that the medium on typical disk drives is partitioned into blocks. In our simulation the block size is 64. Most files contain more bytes than will fit into a single block, so the data of the file are distributed across many blocks. Operating systems control how the data are assigned to blocks, and modern operating systems have elaborate algorithms that attempt to optimize data access against the physical characteristics of the drive itself (rotational latency, head movement, etc.) In our simulation we will implement a very simple algorithm – the data of a file will be stored in consecutive blocks. The data will be taken from the input stream following the create_file command.

Assignment:

(1) Add the following function to the public section of the **diskDrive** class:

- **int findNBlocks(int n, int start)** – finds a sequence of **n** contiguous free blocks on the disk drive occurring at or after block number **start**. The function should return the block number of the first block of the sequence if there is one and –1 if there isn't. This function will be used to find where to store the data of a file. The argument **start** will indicate where the first block after the file descriptor table is.

(2) Augment your function **createFile** to store the actual data of the file. The data will be taken from the next lines of the input. For example, a sequence of lines in the command file might be:

```
create_file pc1 dearmom 65
123456789
123456789
123456789
123456789
123456789
123456789
1234
system_status
```

Note, that <ENTR> counts as a character in a file, just as it would in a text file or a WORD document. Thus, the first line of the file above has 9 readable characters plus the ENTR that terminates the line. Store the data in contiguous blocks of the disk drive, and set the first-block element of the file descriptor to indicate the first block of the file. For example, the above file would require two blocks. If these were blocks 9 and 10, then the file descriptor should have its first block number set to 9. *Add argument containing buffer (main case malloc per file size, read, pass in, then free) or make input file extern.*

(3) Add the following function to the private or protected section of the **computer** class:

- **int findFile(char *fname, fileDescriptor *fd)** – Argument fname is a file name. This function returns the file descriptor number of the file with that name or –1 if the file does not exist in the directory. If the file does exist, the function copies the file descriptor to fd.

(4) Add the following pair of functions to the **computer** class:

- **void printFiles(int argc, char *argv[])** – public function member. Argument **argc** is the token count from the command line for the print_files command, and argument **argv** is the array of parsed tokens. For each file named in argv, if that file exists on this computer, then print it, otherwise print an error message.
- **void printOneFile(fileDescriptor fd)** – protected function member. Argument **fd** is a (copy of a) file descriptor of a file on this computer. This function prints that file.

(5) Implement the PRINT_FILES command in main.cpp. Your case should check to see if the indicated computer exists. If not, print an error message. If it does, pass the number of tokens and the array of tokens into the **printFiles** member function.

Requirements and Specifications:

(1) Your project should process the command list in the file p8input.txt (http://www.eecs.northwestern.edu/~iraicu/teaching/EECS211/code/p8input.txt).

(2) If a request to create a file fails (no file descriptor position available or not enough disk space for the data), print a suitable error message. Note, in this case your program will need to skip past the file contents in p8input.txt to get to the next command.

(3) The command line for the print_files command has the form:

print_files *computer* *file1* *file2* …

where *computer* is the name of a computer in the network and the remaining tokens are file names. If the computer does not exist, your project should print a suitable error message. Otherwise, pass the token count and token array to the **printFiles** function member of the indicated computer. A suitable format for **printOneFile** would be:

Printing file *name*:
text of file, including carriage returns
End of file.

Comments, suggestions, and hints:

The functions from program 7 and this program are designed to make the following algorithm for create_files:

- Find a sequence of blocks of storage for the data.
- If none exists, print error and exit.
- Find an empty file descriptor
- If none exists, print error and exit.
- Create and store the file descriptor.
- Read 64 bytes at a time and store in successive blocks.

The algorithm for finding N consecutive blocks is also relatively simple.

- Set candidate first block to the next free block after the file entry table.
- While not past the end of the disk
    - See if the next N-1 blocks are free
    - If yes, return candidate first block
    - If not, set candidate first block to the next free block after the block that was NOT free.

For example, if the candidate first block was 21 and we needed 5 blocks, but we found block 24 was not free, we would start searching for another free block at 25, the next one after 24.

Remember that the file name from command line is string and needs to be converted to padded 8 characters. Similarly, the byte count is a string and must be converted to integer.

The get() member function of the iostream class returns one character from the input stream, including end-of-line characters. A simple loop using the get() function can be used to read the data for each new file.

See next page for test data.

Test data:

```
add_network_node PC pc1 4096 Larry
add_network_node printer pr1 2048 5
system_status
ls pr1
create_file pc1 f1 20
123456789
abcdefghi
ls pc1
create_file pr1 f1 16
File f1 on pr1.
create_file pc1 f2 70
111111111
222222222
333333333
444444444
555555555
666666666
777777777
create_file pc1 f3 70
aaaaaaaaa
bbbbbbbbb
ccccccccc
ddddddddd
eeeeeeeee
fffffffff
ggggggggg
ls pc1
ls pr1
print_files pr1 f2 f1
print_files pc1 f2 f1 f3
system_status
halt
```