Program 1, TCSS 380 Autumn 2019

OBJECTIVE

The objective of this assignment is to apply your newly learned C knowledge to build a program from scratch.

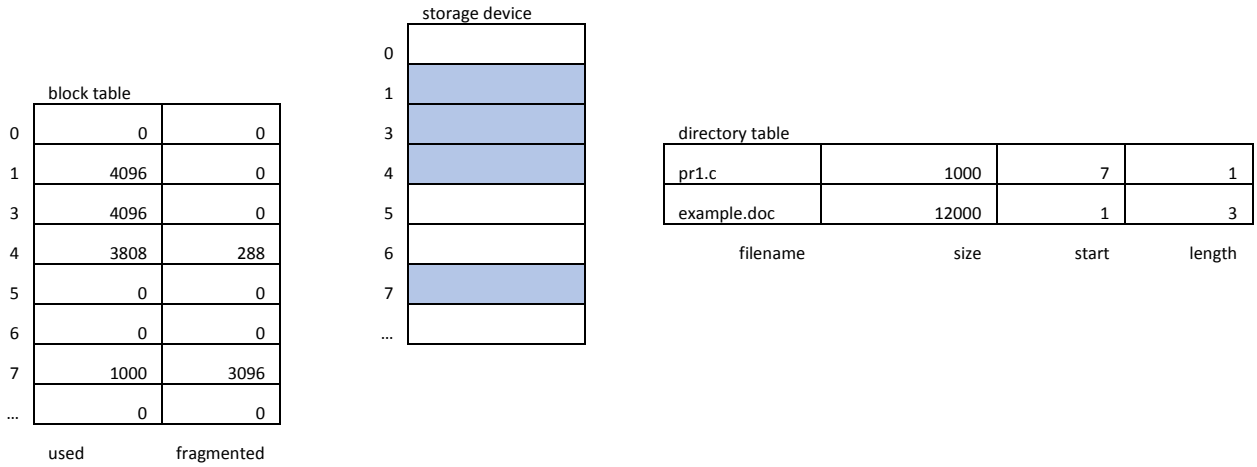
ASSIGNMENT DESCRIPTION

For this assignment you will write a program that models one of the strategies (called contiguous allocation) that an operating system may use to manage the file system space. The general idea for our modeler is as follows: the storage device is a sequence of bytes that is partitioned by the OS into equal size blocks. A file, depending on its size, may be stored in one or more of these blocks. For example, if a block size is 4096 bytes and the file size is 5000 bytes, then the file will occupy 2 blocks: one block will be fully utilized, while the other will be underutilized, with only 904 bytes used. In our modeler, a block cannot be used for more than one file, so in this case we are dealing with internal fragmentation, i.e. underutilization of one specific block by 3192 bytes.

In order to keep track of all the files and where they are located on a storage device, our modeler keeps a couple of tables:

- the directory table that keeps track of all files on the system: name, size, the beginning block, and the number of contiguous blocks each file occupies (in our model files cannot occupy non-contiguous blocks)
- the block table that keeps track of all the blocks: the space actually used by a file within a particular block and the size of internal fragmentation of that block

Each directory entry / file in the program is to be modeled by a structure that consists of a filename and its size in bytes that will be entered by the user, and the number for the beginning block and the number of blocks used by the file that will be generated by the program. The directory table should be modeled by an array of structures. It should be capable of growing as more files are stored on the system.
The block table should be modeled by a 2d array which size is established once the user enters the information for the system. The following diagram shows a conceptual picture of the model:



Your program is responsible for keeping track of the changes to the block and directory tables as files are created and deleted by the user/system. Note that you can model the directory table using a list, whereas the block table is non-resizable but its index could be used as the block number and its availability for storage could be indicated by the *used* component value.

I/O: The input will be provided in an interactive fashion and the output should be generated to the screen. Take a look at a sample run for more details – user input is marked in bold and you should follow the exact same format (e.g. order, menu choices). You can assume that the user of your program will enter valid input, e.g. *storage device size / block size = whole number*, no negative values, no out-of-bounds values, i.e. no error handling of user input required. Filenames will be entered as one word (no blanks in filenames). When prompted for printing, you should

always print an entire block table with all its blocks, whether occupied or not, and all the valid/active files within the directory table.

Enter the size of your storage device: ***40960***
Enter the size of each block: ***4096***
Do you want to:
Add a file? Enter 1
Delete a file? Enter 2
Print values? Enter 3
Quit? Enter 4
1
Adding – enter filename: ***myfile.txt***
Adding - enter file size: ***5000***
Do you want to:
Add a file? Enter 1
Delete a file? Enter 2
Print values? Enter 3
Quit? Enter 4
3
Printing –

Directory table:

Filename	*Size*	*Start*	*Length*
myfile.txt	*5000*	*0*	*2*

Block table:

Block number	*Size used*	*Fragmented*
0	*4096*	*0*
1	*904*	*3192*
2	*0*	
…		
9	*0*	*0*

Do you want to:
Add a file? Enter 1
Delete a file? Enter 2
Print values? Enter 3
Quit? Enter 4
2
Deleting - enter filename: ***mytext.txt***
Do you want to:
Add a process? Enter 1
Delete a process? Enter 2
Print values? Enter 3
Quit? Enter 4
4

PROGRAM SPECS

- The program should follow a multiple file structure (.h and .c files)
- The program should follow ADT principles (cohesion, information hiding, etc.) for the directory table
- The program should employ dynamic allocation for the tables with proper memory management
- You are to provide a driver file that puts everything together – it should be named *driver.c*
- **You need to create a makefile for your code – the code will not be graded otherwise as there are different linking possibilities (should be called makefile)**

- Your executable file should be called *pr1.out*
- All program files must reside in one folder, called *pr1*, and the contents of the entire folder must be compressed using tar utility (instructions below)
- Your tar file should be called *pr1.tar*
- Your program has to follow basic stylistic guidelines, such as proper indentation (it is better to use whitespaces to ensure that's the case), meaningful variable names, etc.
- You need to comment your code
- You are not allowed to use global variables
- **Your program must be compatible with the required Linux VM or the cssgate server – programs that do not compile will receive a grade of 0.** Provide a comment at the top of *driver.c* that specifies which one we should use to run your code.

TAR INSTRUCTIONS

This part is to show you how to compress the entire *pr1* folder. To compress a folder, you have to be in a directory one level higher, i.e. if your *pr1* folder resides in *projects* folder, then to compress *pr1*, you have to be in *projects* folder.

tar -cf <name_of_a_new_tar_file> <directory_to_compress>

tar -cf pr1.tar pr1

To verify your tar file is correct before turning it in, move it to some other directory on your system and decompress

tar -xf <directory_to_decompress>

tar -xf pr1.tar

Verify you see another version of *pr1* folder with all its contents

EXTRA CREDIT

Extend the program in an interesting way or collect stats over multiple runs and add conclusions. For example, you may add error handling to the code: a user enters invalid values or wants to store a file that doesn't fit into the storage space, or wants to delete a file that isn't there, etc. In that case, error messages should be displayed and the program should be able to continue running. Generate stats into a csv file and analyze stats to show whether the block size improves internal or external fragmentation (written as a pdf). If your extra credit results in more code, then you need to submit two files for grading: *pr1.tar* that contains the assignment and *pr1EC.tar* that contains the assignment with extra credit. At the top of *driver.c* explain what your extra credit is.

SUBMISSION AND GRADING NOTES

You are to submit your finished program through *Canvas*. This is an individual assignment – you are NOT allowed to work on it with any other student or share your code with others. The code will be graded based on its correctness by running it on instructor/grader designed test cases. Then, the code will be looked over for anything that violates good coding practices or the assignment specs (e.g. non-meaningful variable names, lack of code modularization). Note that you may turn in this assignment 7 days late, with no penalty. However, no submission past this deadline will be accepted even if you submit one minute after the deadline's timestamp – 7 days of lateness is more than enough.

Grading points will be distributed in the following fashion:

- Running program
  - Directory table operations 15 pts
  - Block table operations 15 pts
  - Memory management 5 pts
- Proper modularization including directory table ADT 10 pts
- Coding style and comments 5 pts
- Extra credit 5 pts