Directory Backup

DUE: MONDAY, November 11, 2013, by 10pm. No late submission accepted.

IMPORTANT: Please read the entire handout before you start coding.

Introduction

Data corruption can cause a significant disruption, which necessitates backing up data. Having learned file operations in C, you are able to write your own program to perform data backup. As an assignment, the ideas used in the program are limited. However, beyond the assignment, it is up to you to expand the ideas to make it applicable to real-world situations, and even to have it run as a daemon to back up a directory automatically.

Project at a Glance

In this project you will write a directory backup program that does the following tasks:

- Keeping track of file changes,
- Mirroring a directory to make a backup,
- Limiting the number of backup instances.

Create an empty file named `backup.c` and write your program inside it from scratch. We provide a tiny header file `backup.h` which defines the default values mentioned in the handout, and you should `#include` it in your program. Besides, you are free to add other headers as long as they are part of C99 standard. You will also get a sample directory for testing.

Note: Using non-standard functions and old interfaces will destroy the portability of your program; it is essential to maintain portability so that your program can run on Autograder.

Please beware that you must NOT use any system calls except for functions for dynamic memory allocation (if necessary) and file operations.

Part 1: The `main()` function

The file `backup.c` should contain the `main()` function. As usual, the `main()` function initializes the memory, parses the command line arguments, and runs the operations accordingly by calling the functions in Parts 2, 3, and 4.

Command-line Arguments

Argument	**Description**
`-s`	Required. This argument specifies the source directory to back up (a.k.a, `sourceDir`).
`-d`	Optional. When present, use the specified directory as the destination to place the backup (a.k.a, `destinationDir`); otherwise, use `DEFAULT_DEST_DIR` by default.
`-m`	Optional. If present, it specifies the maximum number of backup instances to save (a.k.a, `maxBackups`); otherwise, use `DEFAULT_MAX_BACKUPS` by default. E.g., "`-m X`", where `X` is guaranteed to appear, and as a positive integer.

Notes:

1. When the required argument does not show up, or if any of the arguments shows up but does not follow the required format (See the examples below), your program should print a prompt for usage (E.g., "`Usage:\n./backup -s sourceDir [-d destinationDir -m X]\n`") to `stderr`, and exit with `1`.
2. For your reference, the directories specified by "`-s`" or "`-d`" will not have a "`/`" at the end (you don't need to verify this).
   E.g., "`-s /home/cs240/Desktop`" and "`-d /home/cs240/Backup`"
3. The order of the arguments is random. You should not assume orders like "`-s`" as the first one and such like.
4. Some examples of invalid argument lists are:
   a. "`-d /home/cs240/b -m 9`" (missing "`-s`")
   b. "`-s -d /home/cs240 -m 3`" (missing `sourceDir`; applies to "`-d`", too)

E.g., the following command asks your program to back up the given `sourceDir` to the given `destinationDir`, and make sure that there are at most 9 backup instances saved.

```
./backup –s /home/cs240/Desktop -d /home/cs240/MyBackup/ -m 9
```

Part 2: Building and Comparing Logs

To keep track of the changes in `sourceDir`, your program will create and maintain a text log file (whose name is the string constant `LOG_LAST_FILENAME` defined in `backup.h`). It records information of the files and subdirectories inside `sourceDir` from the last run – the file tree structure, the file type, size, time stamp, and so on. Every time your program is run, it should call `createLog()` to create a new log whose file name is defined by `LOG_NEW_FILENAME`, and use `compareLog()` to compare it with the last-run log `LOG_LAST_FILENAME`, if any, and determine whether a new backup needs to be made. After the comparison, replace (update) the file `LOG_LAST_FILENAME` with the new log `LOG_NEW_FILENAME`. *Logs should be stored under `destinationDir` as well.*

By comparing the log files, you will be able to tell if any file has been added, deleted or modified. *HINT: You will need to read the directories recursively to make sure all sub-directories and files are recorded.*

Log format

For each file or directory, the log entry should be of the following format, each entry in one line (symbols "< >" to denote a single field, same use for the next format):

```
<DataType><tab><DataSizeInBytes><tab><CreationTimeStamp><tab>
<LastModificationTimeStamp><tab><DataName><newline>
```

For directory layering, use the following format:

```
<ParentDirectoryInfo><newline>
<tab><SubDirectoryInfo><newline>
```

Note: Files and directories of the same path are aligned, and each sub-directory is indicated with a preceding tab.

You can assume that there are only two types of entries involved – directories and regular files, whose `<DataType>` strings are "`DT_DIR`" and "`DT_REG`", respectively.

Here is an example of a log file for a directory:

```
DT_REG    1         Thu Aug  8 17:52:45 2013    Thu Aug  8 17:52:45 2013    whatever.xxx
DT_REG    1110      Thu Aug  8 17:52:38 2013    Thu Aug  8 17:52:38 2013    music.wav
DT_DIR    4096      Thu Aug  8 17:52:25 2013    Thu Aug  8 17:52:25 2013    engl106
          DT_REG    345       Thu Aug  8 17:52:20 2013    Thu Aug  8 17:52:20 2013    essay1.tex
DT_DIR    4096      Sun Aug 18 14:13:29 2013    Sun Aug 18 14:13:29 2013    cs240
          DT_REG    0         Thu Aug  8 17:52:04 2013    Thu Aug  8 17:52:04 2013    source.h
          DT_REG    12        Sun Aug 18 14:13:27 2013    Sun Aug 18 14:13:27 2013    source.c
          DT_DIR    4096      Thu Aug  8 17:51:49 2013    Thu Aug  8 17:51:49 2013    refs
                    DT_REG    0         Thu Aug  8 17:51:46 2013    Thu Aug  8 17:51:46 2013    handout.pdf
DT_REG    642304    Fri Aug  2 13:34:24 2013    Fri Aug  2 13:34:24 2013    Firefox-latest.exe
```

Implementation

You need to finish the two functions mentioned above for the log part.

```
void createLog(char *sourceDir, char *logFilePath);
```

Create a new log file *for* the directory `sourceDir`, listing the content (subdirectories and files) of `sourceDir` in the specified format. `logFilePath` is a complete path including the file name.

```
int compareLog(FILE *oldLogFile, FILE *newLogFile);
```

Compare the two log files passed in as file pointers. Return `1` if the files are identical; otherwise return `0`.

Part 3: Creating a Backup

If the log files differ, then your program should make a new backup of `sourceDir`.

For every backup, your program should duplicate *all* the content of `sourceDir` to the directory "`destinationDir/YYYY-MM-DD-hh-mm-ss`", where "`YYYY-MM-DD-hh-mm-ss`" is the time when this backup is created. Contents of `sourceDir` should be put directly under this directory (see the example below). To simplify the situation, you can assume that there is no hidden or low permission file in `sourceDir`. To finish this job, two functions are involved: `copyDir()` and `copyFile()`.

Implementation

To do this, you need to implement the function `copyDir()`, which reads the content of `sourceDir` using system call `readdir()` or `scandir()` and copies its structure to the backup directory, and the function `copyFile()`, which merely copies a single, regular file to a given path.

For example, if reading `sourceDir` gives a subdirectory named `dirA`, and `dirA` contains `fileA` and `dirB`, your program should create the corresponding duplicates "`destinationDir/YYYY-MM-DD-hh-mm-ss/dirA/fileA`" and "`destinationDir/YYYY-MM-DD-hh-mm-ss/dirA/dirB`". Please note that *you will need to copy the directories recursively to make sure all sub-directories and files are included.*

`int copyFile(char *sourcePath, char *destinationPath);`
Copy the file (regardless of its format) from `sourcePath` to `destinationPath`. Both paths are complete path with file name. Return `1` if copy is successful; or `0` otherwise.

`int copyDir(char *sourceDir, char *backupDir);`
Copy all the content of `sourceDir` to `backupDir`. Return `1` if copy is successful; or `0` otherwise.

Part 4: Limiting the Number of Backup Instances

After making sure your program can perform backup operations correctly, let's go to the final part: limiting the number of backups. For example, if the maximum number of backup instances to keep (`maxBackups`) is set to be `3`, then only the three most recent backup instances (including the backup to be performed) are kept in `destinationDir`, and older backups will be deleted so that the total number of backups does not exceed `maxBackups`.

Implementation

For this part, you should implement `getNumOfBackup()` by counting the number of directories inside `destinationDir`, and implement `removeOldestBackup()` functions which finds out and deletes the oldest backup. *You will need to delete the directories recursively to make sure all sub-directories and files are removed.*

`int getNumOfBackup(char *destinationDir);`
Return the number of existing backups under `destinationDir`. This parameter equals the one you get from parsing "`-d`" argument.

`int removeOldestBackup(char *destinationDir);`
Remove the oldest backup instance directory under `destinationDir`.

Compilation & Submission

You should be able to come up with the compilation command for the environment you are using. To test your code, please stick to CS240_VM as the environment, and please beware that even if your program can run well under the VM, it may not be portable enough to run correctly on Autograder environment.

To submit your code, enter Autograder and click "Project 3 backup.c". Choose your "`backup.c`" and submit it. You should get your score seconds later provided that grading delay is off.

Grading Criteria

- Source code file `backup.c` reflects good coding style. (25% of total grade)
- Program compiles and runs robustly without problem.
- Logs can be generated in the correct format and content.
- Files and directories can be copied to the destination path correctly and exhaustively.
- *Whenever a file operation fails, print an error prompt and exit the program with 1.*
- The total number of backups can be maintained well.

As always, all projects will be inspected by the instructor and TAs to determine your final grade.