**MCS 253P**

**HW 3**

[Read this first on what to submit for a homework.](https://docs.google.com/document/d/1Gfy08swH0b0HSJ4kXGJLBvraCGz8ywxvmzDQhTQv3QM/edit?usp=sharing)

1. (30 points) make and gdb
   1. Create a subdirectory under your hw directory called hw3 and put all files related to this homework in that directory.
   2. Define a Makefile in hw3 to maintain the program you are writing for this homework assignment (my\_ls). The default target should make my\_ls executable and include gdb debugging information in the executable. Define an additional target called `clean` to remove any files created by a make of your program (e.g., each specific .o file and all the executables - do not use wildcards!). Write your Makefile by hand using vi or vim (same thing). Do not use any of make’s default rules. To test your Makefile, use ‘touch’ to modify various files (like each of the .h files or the .c files), then give a make command to show that only the dependent files are recompiled. The default make should remake your program my\_ls correctly. Divide your program into the following three files: my\_ls.h (declares do\_ls(char \*file)), my\_ls.c (includes my\_ls.h and defined do\_ls()), main.c (includes my\_ls.h and calls do\_ls on appropriate program arguments).
   3. Use the debugger, gdb, while developing your program to help you become more familiar with gdb commands and to help you track down errors. Use the main debugger commands presented in lecture this week. Set breakpoints, set watches (if you can figure them out), run your program with command line arguments within the debugger, single stepping through your program (probably mixing up next and step), printing out the call stack from a breakpoint or from when the program is stopped due to segmentation violation. You may skip trace unless you can figure out how to make it work.

What to submit: connect to your hw3 directory, start a script called hw3.script (by running the Unix command script which records your interactive session with bash - as described in lecture), make clean, make your program, run gdb on your program, set breakpoint (and a watch if you can), run your program with arguments (like a directory to list), when you stop in a break point, print out some variables or parameters, print out the back trace, single step using both step and next, then continue execution until your program finishes, quit gdb, then make clean, cat your Makefile, then exit your script. Submit a zip file of the entire contents of your hw3 directory which includes Makefile, my\_ls.c, hw3.script, but not any .o files or my\_ls executable.

1. (70 points) Write a C program, called my\_ls, that behaves like the Unix command ls -lR (Lowercase L + Uppercase R), e.g.,

$ ls -lR filelist

* 1. Use `readdir` and `stat` to write a C program, called my\_ls, to implement a subset of long, recursive `ls`. Files must be listed one name per line but additionally indented by the nesting depth of the directory. If there are no arguments to my\_ls, list the files in "." (dot) the current working directory. Otherwise, for each command-line argument N, do the following. If N is a directory, print the detailed info for the directory (same as you would for a plain file as below for myFile), then indent 4 spaces and do recursive my\_ls of each entry in N (i.e., like ls -lR, but you **do not need to sort the files**). If N is a file, print the following information for file N in the format (like ls -l):

mode num\_links owner group size\_in\_bytes modification\_timedate name

* 1. Specific Example:

$ my\_ls myFile  
-rw-rw-r-- 1 klefstad faculty 100 Feb 26 07:08 myFile

Explanation of example (see <http://en.wikipedia.org/wiki/Chmod> for mode details)

* -rw-rw-r- mode (file type and access permissions)
* 1 : number of hard-links to this file
* klefstad: owner of the file
* faculty: group of the file
* 100: size of the file in bytes
* Feb 26 07:08 last modification date and time
* myFile: file/directory name
  1. You need not handle symbolic links or devices (just files and directories). Do not recurse into “.” or “..” which are in each directory. Here are some snippets of code for inspiration (but don’t copy and paste). One reads the files in a specified directory. The other converts a user ID integer into a symbolic string. Note **DO NOT** use “[scandir](http://man7.org/linux/man-pages/man3/scandir.3.html)” which is a C-lib function that reads a directory, but will leak memory unless you free each file name. Print the mode using the same symbolic format used by ls. Print the user name as a symbolic string, like klefstad, not as an integer.

#include <sys/types.h>  
#include <dirent.h>  
#include <errno.h>  
int do\_ls(char \*dirName)

{

DIR \*dirp;  
 dirent \*direntp;  
 if (!(dirp = opendir(dirName)))

{  
 fprintf(, stderr, "Error(%d) opening %s\n", errno, dirName);  
 return errno;  
 }  
 while ((direntp = readdir(dirp)))

printf(“%s\n”, direntp->d\_name); /\* use stat here to find attributes of file \*/

closedir(dirp);

}

char \*getUserName(uid\_t uid) /\* how to convert a user ID to a user name \*/

{

struct passwd \*pw = getpwuid(uid);

return pw->pw\_name ? pw->pw\_name : "";

}

How to convert a Linux stat time to a string. Check out the use of sprintf.

<http://stackoverflow.com/questions/13542345/how-to-convert-st-mtime-which-get-from-stat-function-to-string-or-char>

**What to put in your report:**

**Problem 1**: Paste the session that you recorded (hw3.script).

**Problem 2**: Code for important functions like the recursive function and the function that prints the information about the files. Plus a screenshot of execution of my\_ls (entire execution not needed for screenshot, just first few lines that fits on the screen).

**Submit two things:**

1. **your report (report.pdf - if you do a text file, print to pdf). Your report should contain sections for each problem above. Submit reports for each question separately on gradescope.**
2. **your files (Makefile, my\_ls.c, my\_ls.h, main.c) zipped into a file named hw3.zip on Canvas.**

**A good test case for my\_ls is /usr/local/ on openlab.ics.uci.edu. The output of "ls -lR /usr/local " is relatively short - about 134 lines. There are empty subdirectories so don’t be confused by them.**