CPSC 457 - Assignment 2

Due date: **Friday, February 15, 2019 at 11:30pm**.
Individual assignment. No group work allowed.
Weight: 6% of the final grade.

Please visit the main assignment page for additional hints for the questions below.

Q1 – Bash programming question (10 marks)

Write a bash shell script "`scan.sh`" that takes two arguments:

1. suffix – a string
2. N – a positive integer

Your shell script will recursively scan the current directory for all files ending with the given suffix, then sort the files by their size, and then print the the filenames of the largest N files to the standard output, followed by the sum of their sizes. Each file reported should be followed by the size of the file in bytes. If the total number of files found is less then N, report all files. The listed files should be sorted by the size, in descending order.

Sample output:

```
$ scan.sh jpg 5
./face.jpg 88374
./a/b/school-jpg 88339
./a/tree.jpg 38883
./b/housejpg 3001
./a/b/jpg 233
Total size: 218830
```

Hints:

- You may consult the manual pages for one or more of the following utilities: `find, sort, head, awk`
- You may use pipes "`|`" to feed the output of one utility program into another utility program as input.
- You can assume there will be no spaces in any of the filenames or directory names.

Q2 – C/C++ programming question (14 marks)

Write a C or C++ program called `myFind.c` or `myFind.cpp` which replicates a small subset of the functionality of the `find` system utility. In particular, the output of your program should be identical to executing the following:

```
$ find . -type f
```

The above command recusively finds all files in the current directory and prints them one line at a time to the standard output. In your implementation you may use the following system calls:

- `opendir(), closedir(), readdir()`
- `printf() and/or std::cout`

Your program should take no command line arguments. If you are confortable with recursion, you may mplement a recursive solution, and if not, you can implement an iterative solution using a stack. The order of the filenames reported by your program does not have to be identical to that of find, but it has to report the same set of filenames.

Q3 – Programming question (15 marks)

Implement Q1 in C or C++, and name your program `scan.c` or `scan.cpp`. Your program should make a one time use of the `popen()` functions to invoke your solution to Q2 as an external command – to get a recursive list of files in the current directory, eg.:

```
FILE * fp = popen("./myFind", "r");
```

If you failed to implement Q2 correctly, you may use the following instead:

```
FILE * fp = popen("find . -type f", "r");
```

All other operations **must** be performed in C/C++, such as:

- determining the file size,
- filtering filenames based on the extension,
- sorting,
- calculating the sum.

In addition to `popen()`, you may find it useful to use the following functions:

- `stat()` to determine the size of the file;
- `qsort()` for C or `std::sort()` for C++, for sorting purposes.

You may assume that:

- the total number of files in the directory will be less than 1000, and
- the maximum pathname length of any file will be less than 512.

The output of your C/C++ program should be identical to the output of your bash script from Q1.

Q4 – Written question (3 marks)

Run "`strace -c`" and "`time`" on your bash script from Q1 and your C/C++ program from Q3 and compare the results. Include the output of the above commands in your report, and explain why the results are different.

Q5 – Programming question (10 marks)

Write a multi-threaded solution for computing the sum of a large array of integers. Your program, "`sum.c`" or "`sum.cpp`", will take two arguments from the command line. The first argument is the name of the file containing the integers (one number per line) to be read into the array, and the second argument, T, is the number of threads to be created. You may assume that the input contains N integers where N ≤ 1,000,000 and the number of threads is always smaller than or equal to the number of integers in the file, i.e. T ≤ N.

Your program will divide the array into T groups of roughly equal size:

- the first $(N\%T)$ groups will contain $(N/T + 1)$ number of elements;
- the remaining $(T - N\%T)$ groups will contain (N/T) elements.

For example, given T=2 and the array [4,7,3,5,98,23,53], the first group will contain [4,7,3,5] and the second group [98,23,53]. Each thread will compute the sum of the integers in one of the groups.Your program should display the sum of the integers assigned to each thread and finally the overall sum. The expected output for the above example is as follows, assuming the numbers are provided in `input.txt`:

```
$./sum input.txt 2
Thread 1: 19
Thread 2: 174
Sum = 193
```

The numbers after the word "Thread" in each line are the thread IDs. During the marking session your TAs will run your code on different input files and with different thread numbers, so it is important that your program accepts command-line arguments instead of hardcoded input file name and the number T.

Submission

You should submit 5 files for this assignment:

- Answers to the written questions combined into a single file, called either `report.txt` or `report.pdf`. Do not use any other file formats.
- Your solutions to Q1, Q2, Q3 and Q5 in files called `scan.sh, myFind.c, scan.c` and `sum.c.`

Since D2L will be configured to accept only a single file, you will need to submit an archive, eg. `a2.tgz`. To create such an archive, you could use a command similar to this:

```
tar czvf a2.tgz report.pdf scan.sh myFind.cpp scan.c sum.cpp
```

General information about all assignments:

1. All assignments are due on the date listed on the assignment. Late assignments or components of assignments will not be accepted for marking without approval for an extension beforehand. What you have submitted in D2L as of the due date is what will be marked.
2. Extensions may be granted for reasonable cases, but only by the course instructor, and only with the receipt of the appropriate documentation (e.g. a doctor's note). Typical examples of reasonable cases for an extension include: illness or a death in the family. Cases where extensions will not be granted include situations that are typical of student life, such as having multiple due dates, work commitments, etc. Forgetting to hand in your assignment on time is not a valid reason for getting an extension.
3. After you submit your work to D2L, make sure that you check the content of your submission. It's your responsibility to do this, so make sure that you submit your assignment with enough time before it is due so that you can double-check your upload, and possibly re-upload the assignment.

4. All assignments should include contact information, including full name, student ID and tutorial section, at the very top of each file submitted.
5. Assignments must reflect individual work. Group work is not allowed in this class nor can you copy the work of others. For further information on plagiarism, cheating and other academic misconduct, check the information at this link: http://www.ucalgary.ca/pubs/calendar/current/k-5.html.
6. You can and should submit many times before the due date. D2L will simply overwrite previous submissions with newer ones. It's better to submit incomplete work for a chance of getting partial marks, than not to submit anything.
7. Only one file can be submitted per assignment. If you need to submit multiple files, you can put them into a single container. The container types supported will be ZIP and TAR. No other formats will be accepted.
8. Assignments will be marked by your TAs. If you have questions about assignment marking, contact your TA first. If you still have questions after you have talked to your TA then you can contact your instructor.

Appendix 1 – Pseudocode for Q2

Pseudocode for recursive algorithm for finding all files:

```
find( dirpath):
  open directory dirpath
  for every entry e in the directory:
    fullpath = dirpath + "/" + filename of e
    if entry e is a file
      print fullpath
    if e is a directory
      find(fullpath)
  close directory

find(".")
```

Pseudocode for iterative algorithm for finding all files:

```
create empty stack s
s.push(".")
while s is not empty
  dirpath = s.pop()
  open directory dirpath
  for every entry e in the directory:
    fullpath = dirpath + "/" + filename of e
    if entry e is a file
      print fullpath
    if e is a directory
      s.push(fullpath)
  close directory
```