Operating Systems (CS:3620)

Assignment 3   Total points: 100

Due: by 11:59 pm of September 24, 2017

This assignment will contribute 5% to your final grades.

Submission: Please submit your source code through ICON. Each problem should be placed in it's specific directory. For instance Problem 1 should be in the Problem 1 folder. All of these folders should be in a single folder named "Assignment-3". You will then zip the "Assignment-3" directory and will name the zip file in the following way: *<last_name>-<first_name>-Assignment_3.zip*. You should replace <last_name> and <first_name> with your last and first name, respectively. For example, if a student's name is Bob Lee the submission zip file should be called `Lee-Bob-Assignment_3.zip`.

Instructions: For each of the following problems, you will need to write a C file to solve each problem. The name of each C file for each problem is mentioned in the associated problem statement below. For instance, the name of the C file for Problem 1 is "fastsort.c" Please refer back to the files found in ICON under the "Supplementary Material" folder for help.

Please make sure your source code compiles with GCC, otherwise you will not obtain any points.

For compiling a C program named `problem1.c` in the terminal using `gcc` and in the process generate the executable file `problem1`, type in the following command: `gcc Problem1.c -o problem1`. To run the executable without any command line arguments, you would need to type in `./problem1` in the terminal.

Grading: We will use automated grading scripts with pre-determined inputs and some manual inspection to check each program for correctness. *Please make sure to follow the C file name convention for each problem exactly as directed.* Otherwise, your assignment may not be graded properly. Please make sure you are using an Ubuntu environment for this assignment. We will provide a Ubuntu virtual machine image. Please see the end of the assignment for detailed instructions about setting up the virtual machine. Note: This is the same image as Assignment 1. **If you successfully installed this image, there is no need to go through the installation process again.**

Cheating and Collaboration: ***This is an individual project****, you can discuss with your peers but cannot copy source code. Please do not copy source code from Internet. Think about the worst-case scenario when you get caught.*

Questions

1. (50 points) **Binary Record Sorting in Ascending Order**

   This program should be called `fastsort.c`. It should be placed inside the "Problem 1" directory.

   You will have to write a sorting program. Your program should read in a file which has the following format

   ```
   <4 byte key><96 byte value>
   <4 byte key><96 byte value>
   ```

   You will have to build a sorting program called fastsort that takes in one of these generated files and sorts it based on the 4-byte key (the remainder of the record should of course be kept with the same key). The output is written to the specified output file.

   Generating Input Files

   To generate test files we have provided a simple program called `generate`. This program will generate input files to test your program.

   Using generate is simple. First you should compile it with the following command:

   ```
   shell% gcc -o generate generate.c -Wall -Werror
   ```

Note: you will also need the header file sort.h to compile this program.

Once you have successfully compiled the generate.c file with the command above, run the following command to run it:

```
shell% ./generate -s 0 -n 100 -o /tmp/outfile
```

Note: 0, 100 and /tmp/outfile are examples of arguments passed to generate.

There are three flags to generate that need an argument to go with it. The **-s** flag specifies a random seed for the random number generator; this allows you to generate different input values to test your sorting program. The **-n** flag determines how many records you want to generate in your input file, each of size 100 bytes. Finally, the **-o** flag determines the name of your input file, which will be the input file for your sorting program.

The format of the file generated by the generate.c program is very simple: it is in binary form, and consists of the 100-byte records as described above. A common header file sort.h has the detailed description.

Debugging Your Sorting Algorithm

Another useful tool is *dump.c* which is provided as well. This program can be used to dump the contents of a file generated by generate or by your sorting program to the standard output.

You can compile it with the following command:

```
shell% gcc -o dump dump.c
```

Once you have compiled the dump.c file you can run it with the following command:

```
shell% ./dump -i /tmp/outfile
```

There is only one flag when using dump. The **-i** flag specifies the name of the input binary file.

Hints: In your sorting program, you should just use `open()`, `read()`, `write()`, and `close()` to access and write to the input/output binary files. See the code in generate or dump for examples.

If you want to figure out how big in the input file is before reading it in, use the `stat()` or `fstat()` calls.

To exit your program call `exit()` with a single argument. This argument to `exit()` is then available to the user to see if the program returned an error (i.e., return 1 by calling `exit(1)`) or exited cleanly (i.e., returned 0 by calling `exit(0)`).

The routine `malloc()` is useful for dynamic memory allocation. Make sure to exit cleanly if `malloc` fails!

If you don't know how to use these functions, use the man pages. For example, typing `man malloc` will provide you information about how to use the malloc function.

Assumptions and Errors:

32-bit integer range. You may assume that the keys are unsigned 32-bit integers.

File length: May be pretty long! However, there is no need to implement a fancy two-pass sort or anything like that; the data set will fit into memory.

Invalid files: If the user specifies an input or output file that you cannot open (for whatever reason), the sort should EXACTLY print: **Error: Cannot open file foo**, with no extra spaces (if the file was named foo) and then exit.

Too few or many arguments passed to program: If the user runs fastsort without any arguments, or in some other way passes incorrect flags and such to fastsort, print **Usage: fastsort -i inputfile -o outputfile**, with no extra spaces and then exit.

Important: On any error code, you should print the error to the screen using fprintf() , and send the error message to stderr (standard error) and not stdout (standard output). This is accomplished in your C code as follows:

```
fprintf(stderr, ‘‘whatever the error message is\n’’);
```

2. (50 points) **Toy UNIX shell:**

This program should be called `mysh.c`. It should be placed inside the “Problem 2” directory.

For this problem you will be writing a small toy UNIX shell. This shell will be very basic and only support some basic commands.

Your shell essentially should be an interactive loop that prints **mysh** >. The program should parse the input and execute the given command. It should continue until the **exit** command is executed in which case your toy shell should exit.

Commands

`pwd` - This command should display the current directory. **Hint: You can use `getcwd()` to get the current working directory.**

`cd` - This command should take you to your home directory. You can use `getenv("HOME")` to get this. **Hint: You can use chdir() to change the current working directory.**

`cd [one or more spaces] dir` - This command should change the current directory to the `dir` provided as an argument to cd. **Hint: You can use chdir() to change the current working directory.**

`show-dirs` - This command should print all the sub-directories in the **current** directory. The directories should be printed out one per line. **Hint: You can use `readdir()` to get sub-directories and files from the given directory.**

Note: Order does not matter when printing the directories names to the standard output.

`show-files` - This command should print all the files in the **current** directory. The files should be printed out one per line. **Hint: You can use `readdir()` to get sub-directories and files from the given directory.**

Note: Order does not matter when printing the file names to the standard output.

`mkdir [one or more spaces] dir` - This command should make a directory with the given `dir` name. **Note:** If a directory with that name already exists you should print the following error to the console. < *dirname* > *already exists.*

Note: The given `dir` name will **NOT** contain any spaces.

Hint: You can use `mkdir()` to create a new directory.

`touch [one or more spaces] file` - This command should create an empty file in the current working directory with the filename provided. **Note:** The given `file` name will **NOT** contain any spaces.

`clear` - This command should clear the terminal. **Hint: Look at how an actual UNIX shell does this.**

`exit` - This command should exit the program and close the toy UNIX shell

For instance, the following example is what your shell should look like.

Note: This example starts by executing your file with ./mysh since in this example the c file was compiled using the following command, gcc mysh.c -o mysh

```
prompt > ./mysh
mysh > pwd
/home/mitziu/Documents/OS2017/Assignment_3
mysh > cd ..
mysh > pwd
```

```
/home/mitziu/Document/OS2017
mysh > show-dirs
Assignment_1
Assignment_2
Assignment_3
.
..
mysh > mkdir temp_dir
mysh > mkdir temp_dir
temp_dir already exists.
mysh > show-dirs
Assignment_1
Assignment_2
Assignment_3
.
..
temp_dir
mysh > cd temp_dir
mysh > pwd
/home/mitziu/Documents/OS2017/temp_dir
mysh > show-files
mysh > touch temp.txt
mysh > show-files
temp.txt
mysh > show-dirs
.
..
mysh > cd
mysh > pwd
/home/mitziu
mysh > exit
```

Virtual Machine Instructions:

For this assignment an Ubuntu image has been provided. Please see “**Instructions**” for links and instructions on how to set everything up.

Instructions

1. Please download and install **VirtualBox** from the following link:
   https://www.virtualbox.org/wiki/Downloads

2. Please download the **sec.ova** images from the following link:
   `https://www.dropbox.com/sh/hpgt3tuyev8hlt6/AADkefjEtdzp9mP4YXU1DvFNa?dl=0`

3. Run **VirtualBox**

4. Go to **File/Import Appliance**

5. In the *Appliance to import* window choose *sec.ova* image file and press **Continue**

6. Press **Import** and then the import process will begin

7. After the *Import* process finishes you can see the imported virtual machine (VM) on the left-side window. Select the VM and then press **Start** in the toolbar.

8. Whenever prompted for username and password use the following credential
   - **Username:** sec
   - **Passowrd:** ces
9. After logging in. Go to ICON from the VM and download Assignment 3 found in ICON.