CS3411 Project 4 - s-tee

Due : Thursday, Nov 29, 2018 AoE

In this project, you will be developing a *s-tee* program, similar to Unix tee program that branches a data stream and sends it to two destinations, namely, the standard output and a file. However, our s-tee program will be different. Unix tee program reads its standard input and writes the input to both the standard output and the passed file argument. Instead, our program will accept the output file name as the first argument which will be followed by the program to be executed and its arguments. The s-tee program then first forks off a child and establishes a pipe between the child and itself by changing its standard output to the pipe. The child should read from the pipe, write the data that it read to its standard output and the passed file argument. Once the pipe communication is established, **the parent first writes all of its arguments verbatim to its standard output followed by a newline character**. Next, it executes an *exec* kernel call to overrite itself with the program to be executed. Note that the child should not read from the standard input and **everything the parent writes should go to the pipe, including data written to both its standard error and standard output**.

The syntax for the execution of the *s-tee* program is given by:

```
s-tee <file name> <program name> <arguments>
```

where the `<file name>` is any valid filename, `<program name>` is the full path to the to be executed program, `<arguments>` are the arguments which need to be passed to that program.

```
s-tee my-directory-listing /usr/bin/ls /home/myuserid
```

executes /usr/bin/ls with the argument /home/myuserid and **creates a file** titled my-directory-listing where the file listing that was seen on the screen would also appear.

Requirements

1. If the argument file already exists, give an error and exit.
2. The program should properly print an error message arising from ALL kernel calls, such as open, read, write. **Any error should cause the termination of all processes**.
3. The program MUST not leave hung process(es) behind.
4. The program MUST use pipes to communicate and MUST obey the communication structure outlined above.
5. Use of anything other than kernel read/write calls for I/O disqualifies the program.
6. Use a buffer size of 1024 bytes to read data from the pipe and write as many as read from the pipe to the output file and the standard output.
7. Do not attempt to verify the program argument to see if t is executable. Perform the exec call and if it fails, **bail out properly** after issuing an error message (e.g., *s-tee bogus* should fail if there is no executable called *bogus*).

Your submission should include a tarred gzipped copy of :

1. Your program source file.
2. A README file.
3. A Makefile.

Your makefile should include "all", and "clean" labels. When I type `make` or `make all` in the directory containing your recovered submission, a binary file named `s-tee` should be created. Typing `make clean` should remove all the object files and the created binary `s-tee`. Projects will be graded on the lab Linux machines.

Ground Rules and Restrictions This assignment may be provided with updates as the project goes on. Make sure that you attend the classes and watch the class mailing list.
You may not borrow or reuse any code from other resources, and all the code must be your own. In addition, the following rules apply:

1. You may discuss the program with others.
2. You may not bring any printed/written material into the discussion with you. (You may not show anyone your code; you may not view the code of anyone else. This includes others both enrolled in the course and others not enrolled in the course.)
3. You may generate written material during the discussion, but it must be destroyed immediately afterwards. Don't carry anything away from the discussion.
4. If you are in doubt about the acceptability of any collaboration activity, I expect you to check with me before engaging in the activity.