CS-3411 Program IV : Husky Script

Fall 2023

In this project, you will develop a *script* program called *husky script* or simply `hscript`. Similar to the `script` command, `hscript` forks and executes some program and intercepts all input and output to and from that program. The intercepted data is sent to the terminal to appear as normal but it is also saved to files within a specified directory. In this way, when running a program using `hscript`, the experience will appear just as if the program is running normally, but with the additional functionality of having the input and output logged.

Syntax

The syntax for `hscript` is as follows:

```
./hscript <program name> <arguments> <directory>
```

- `<program name>` is the full or relative path of the program to be executed.
- `<arguments>` are a list of zero or more arguments that are passed to the program to be executed.
- `<directory>` is the local directory where the intercepted data is written to.

Take the `ls` program as an example:

```
./hscript ls -l -a -h test
```

Here, the command `ls -l -a -h` is to be executed, and the input and output intercepted and saved to files within the `test` directory.

Requirements

Here is a list of additional requirements:

1. You must write the program in C.
2. If the specified directory (`<directory>`) does not exist, the program should create it with the appropriate permissions. If the program runs into an error when trying to create the directory (for example, if there is already a file with that name) treat this as any other error.
3. Intercept stdin (fd 0). The stdin of `hscript` should be sent to the stdin of the child program and to a file in the specified directory named "0".
4. Intercept stdout and stderr (fd 1 and fd 2). The outputs of the child program should be redirected to `hscript`. It is then the job of `hscript` to output this to its own stdout and stderr as well as to files in the specified directory named "1" and "2".
5. The parent and child communication necessary for interception should be done using pipes.
6. Create and truncate the files "0", "1", and "2" in the specified directory and do so with the appropriate permissions.
7. Use appropriately sized read and write buffers (e.g. 1024 bytes) for file and pipe input/output.
8. The program should never block (e.g. waiting on a `read()` of a pipe). Use the `select()` system call and ensure the pipes are correctly configured.
9. Input/output redirection should work as normal. For example, `./hscript cat test < infile > outfile` should make a copy of `infile` called `outfile`. The files `./test/0` and `./test/1` should then be another copy of `infile` and `./test/2` should be empty.

10. If any error is detected, the program should print an appropriate error message and exit.

11. Do not have any GCC warnings when compiling with -Wall and -Wextra. Let us know if you have any "unavoidable" warnings; they may be excused.

12. Create a Makefile that includes the *all* and *clean* labels as with previous programs. Calling either `make` or `make all` in the submission directory should create a binary file named `hscript`.

13. Use system calls in all cases where a system call alternative exists. For this program you have the following libraries, functions, and system calls in your toolbox: `string.h`, `signal.h`, `malloc()`, `free()`, `open()`, `read()`, `write()`, `select()`, `_exit()`, `sprintf()`, `waitpid()`, `fork()`, `execvp()`, `pipe()`, `pipe2()`, `dup()`, `dup2()`, `close()`, `stat()`, `mkdir()`. You may use others additionally, but you should not necessarily need to.

Submission

Include and submit the following to Canvas in a zip file:

1. The program source file(s).
2. A Makefile.