Operating System Fundamentals: Assignment 1

Lyndon Renaud 104 566 776

Sample input/output

Case 1: input file exists and is readable, output file does not exist

```
$ ls .
  a1.c   input.txt
$ clang a1.c -o a1
$ ./a1
  Enter input file name: input.txt
  Enter output file name: output.txt
  File copy complete!
$ ls .
  a1.c   a1   input.txt   output.txt
```

Case 2: input file exists and is readable, output file exists

```
$ ls .
  a1.c   input.txt   output.txt
$ clang a1.c -o a1
$ ./a1
  Enter input file name: input.txt
  Enter output file name: output.txt
  Output file already exists. It cannot be overwritten. Aborting.
$ ls .
  a1.c   a1   input.txt   output.txt
```

Case 3: input file does not exist or is not readable

```
$ ls.
  a1.c
$ clang a1.c -o a1
$ ./a1
  Enter input file name: input.txt
  Enter output file name: output.txt
  Input file either does not exist or cannot be read. Aborting.
$ ls .
  a1.c   a1
```

Implementation Details

1. Program setup

We must ask the user for two files. These files have names of variable length, but we know the max path length in an EXT4 filesystem is 4096 characters. We use this value to set the maximum length of the char arrays storing the file names. We are also going to do a lot of reading/writing byte by byte, so I declared a single char buffer, `buf`.

2. Prompt for input file

The first task is to prompt the user for an input file. We write a char array containing the prompt message to the standard output using `write()`.

3. Read input file name

The user will type in their desired file and then hit `<Enter>`. We assume everything before the `\n` character is part of the filename and use `read()` to read in characters one by one. Once `\n` is encountered, a `\0` is appended to the string and the loop breaks.

4. Prompt for output file

We repeat the input file prompt process, but this time for the output file.

5. Read output file name

Similar to how we read the input file name, we read the output file name.

6. Check if input file exists and open

I used `access()` to check if the input file exists and if we can even read its contents. If the file doesn't pass the test, the program displays an error message using `write()` and terminates with `_exit(1)`. Else, a file descriptor `input_fd` for the input file is made using `open()` in read only mode.

7. Check if output file exists

I used `access()` to check if the output file already exists. If it does, the program displays an error message using `write()` and terminates with `_exit(1)`. Else, a file descriptor for the output file is created using `open()` in write only mode.

8. Copy the file byte by byte

In a loop, this copies every input byte to the single byte buffer and then writes it to the output file descriptor.

9. Close the file descriptors

Make sure to close any open file descriptors

10. Complete and exit

Print a message to the standard output denoting completion, then terminate with `_exit(0)`.