CSE 325 - Project 2

Project Overview

This project focuses on system-level programming in a Linux environment. With the aid of system calls, you will design and implement the C++ program which copies the contents of one file to another file, as described below. You will complete the exercises below and submit your work for grading via the D2L system.

Project Deliverable

The deliverable for this project is the following files:

proj02.cpp – your program code

Make sure you are submitting the correct file names.

Project Specifications

1. The purpose of the program is to copy the contents of one file (the source file) to another file (the destination file). The user will interact with the program through command line arguments.

   - The program will process the command line arguments from left to right, and it will process all command line arguments before manipulating the source file or the destination file.
   - If an argument begins with the character '-' (normal dash), it will be processed as an option which controls the behavior of the program. Valid options are "-b", "-a" and "-t" (defined below).
   - If an argument does not begin with the character '-', it will be processed as a file name. Exactly two file names must be provided by the user, with the source file listed first and the destination file listed second.

2. The program will use the following functions to manage the files:

   int open(const char *pathname, int flags, mode_t mode);

   int close(int fd);

3. The program will use the following functions to perform input and output operations on the files:

   ssize_t read(int fd, void *buf, size_t count);

   ssize_t write(int fd, const void *buf, size_t count);

The size of the buffers associated with those functions will be determined by the "-b" option.

4. The program will recognize the following options:

   - The option "-b" will be followed by a separate command-line argument which indicates the size of the buffer (in bytes) to be used while processing the source file and the destination file. The default buffer size will be 64 bytes.
   - The option "-a" option will cause the program to append the source file to the end of an existing destination file.
   - The option "-t" option will cause the program to truncate an existing destination file, and then copy the source file into the destination file.
   - If neither the option "-a" nor the option "-t" is selected by the user, an existing destination file will not be altered.
   - The options "-a", and "-t" will have no impact on a destination file which does not exist when the program begins execution (the program will always attempt to create a destination file which does not exist).

5. The program will minimize the number of calls to function "write", within the constraints imposed by the size of the buffer.

6. The program will include appropriate logic to handle exceptional cases and errors. You are welcome to publicly discuss edge cases logic on Ed Discussion without sharing your solution code publicly.

Project Notes

1. As stated above, your source code file will be named "proj02.cpp" and you must use "g++" to translate your source code file in the *scully* Linux environment.

2. The purpose of the program is to copy the contents of the source file to the destination file using the "read()" and "write()" system calls. Consider the following example, where "infile" contains 150 bytes and "outfile" does not exist:

   proj02 infile outfile

   Since no program options are specified, the defaults will be used (including a buffer size of 64 bytes). The program will call function "read()" four times and function "write()" three times to transfer the 150 bytes from "infile" to "outfile" (reading and writing 64 bytes, reading and writing 64 bytes, reading and writing 22 bytes, reading 0 bytes).

3. As stated above, the command line arguments will be processed from left to right. Consider the following examples, where "fileA" and "fileB" both exist and can be accessed by the user:

   proj02 fileA -a fileB -b 256

   proj02 –b 256 -a fileA fileB

   proj02 fileA fileB –b 256 -a

   All three commands will have the same effect: the program will use a buffer size of 256 bytes, and will append the contents of the source file ("fileA") to the end of the destination file ("fileB").

4. Each command line argument is constructed as a low-level character string (array of type "char", with a terminating null byte). If you prefer to process a command line argument as a C++ string class object, you should consider converting the low-level character string. For example:

   string prog = string(argv[0]);

5. The second argument to function "open" allows you to control the behavior of that function. Some combination of one or more of the following might be useful: **O_RDONLY, O_WRONLY, O_CREAT, O_EXCL, O_APPEND, O_TRUNC**.

6. The third argument to function "open" allows you to control the file permissions associated with a file which is being created. You would be wise to use S_IRUSR — S_IWUSR (or the equivalent) so that you can examine the contents of any new files which your program creates. Otherwise, you will have to use the "chmod" command with each file to change the file permissions for that file.

Additional Notes

- Any detail not covered in the project specification is up to you to decide. For example, in such cases you may choose to recover from errors or simply terminate the program. Our main advice is to think about how you would like the program to behave if you were the user: what kind of information you would want it to show, and how you would want it to respond when an error occurs or when invalid input is given.