**Myls**

**Mycat**

The Linux cat command is a very simple piece of functionality that is designed to do one thing very well - that is, displaying the contents of a file to the terminal, and concatenating multiple files in the case of multiple arguments. Additionally, cat may operate with zero arguments, in which case it will take its input from the keyboard. cat may also create new files, append to existing files, and provide input to subsequent commands by using the '>', '>>', and '|' operators, respectively. The core of our implementation involves iterating through each argument passed to the function. For each argument, a number of error checks are performed, and the file is then opened and read character by character, printing each character to the standard output stream until the end of file condition is reached. The file stream is then closed, and the program iterates to the next file to be read, or terminates if there are none remaining. In the case that no arguments are passed to the function, the program allocates a 256 byte character buffer and reads from standard input line by line using the getline() method. This continues until it is interrupted by a signal from the terminal.

The most basic error occurs when a specified file is not found in the provided location. In this case, the fopen() function will return a NULL value to our file pointer, which signals the process to print an error message and continue to the next file in sequence. Another potential issue that our program must check for is the type of file provided by the user. Specifically, if the file is a directory, the cat command will not attempt to open a stream or read from it. While a directory is indeed a file containing a list of contents and their inode numbers, it is not intended to be read from in most cases. Our program uses the stat functionality in order to read in a structure of file features, allowing us to confirm if the file is a directory. If this is the case, the program will not terminate, but rather display an error message to the terminal and iterate to the next argument.

**Mycp**

Functionality

This program emulates the actions of the cp shell command. The program’s file copy functionality expects two arguments in the form './mycp.exe a\_file b\_file' it will then copy the contents of the text file "a\_file" to the file "b\_file". The program will create b\_file in the current directory, if it has not yet been created. The program will prompt user for permission if b\_file is to be overwritten.

The programs directory copy functionality expects arguments and option in the form './mycp.exe -R a\_dir b\_dir`. The function will then traverse the directory structure recursively copying all sub folders/directories. The program expects rwx permission on the two directories.

Expected Conditions

The user shall not pass command arguments in form other than (mycp.exe a\_file b\_file and mycp.exe -R a\_dir b\_dir ). a\_file and a\_dir must exist before the function is called. a\_file is expected to have “r” permission while a\_dir is expected to have “wx”. b\_file does not have to previously exist, however if it does it must have “w” permission enabled. Simillarly b\_dir does not have to previously exist, however if it does it must have “rx” permissions.

Implementation

The copy file functionality opens both files and copies their contents line by line. The b\_file will be created if not already. The function will fail if a\_file doesn’t have read permission. The amount of data to be read in per line is an limitation, see Limitation section for further detail.

The cpdir function uses a while loop read in files contained in a\_dir and then create them in the b\_dir. The contents of the files are then copied over one character at a time. If the object read from a\_dir is its self a directory the cpdir function is called recursively with the new directory pointer passed to it. The opendir() and readdir() functions are used to work with the directory structures.

Limitations

The fgets() function used to copy files line by line is only allotted 1100 character spaces to allocate the information. This is larger than the max line size in my system, however if moved across systems user should be aware of this limitation. The cpdir() function saves the current path of the program and the relative path of the a\_dir directory’s tree structure. The total amount of space allocated for this is 256 characters. My current system has a max path length of 246 characters. If the program is moved across systems user should be aware of this limitation.