```
//ICSI 333. System Fundamentals
//Peter Buonaiuto: 001497626
//TA Omkar Kulkarni
//Spring 2022

/*
This program will take a user command in the form of one of the following:
    ./copy  source1 [source2 ...] dest
    ./move source1 [source2 ...] dest

    Despite the mode, it will attempt to copy all files to the given destination (or device, if
    copying). Fatal errors may be thrown within runtime if there is no chance for one
    successful iteration
    and in this case the main will return with EXIT_FAILURE. Non fatal warnings will be
    displayed at the end, as will a synopsis of successful moves that have occured.

    The program will unlink the original sources of copied files if we are in move mode.
    The porgram can accept relative OR complete paths as arguments.

    Compile command:
        gcc project3.c -o copy; ln copy move

    Copy one or two files
        ./copy file1.rtf CopyTest/
        ./copy file1.rtf file2.rtf CopyTest/

    Move one or two files
        ./move file1.rtf CopyTest/
        ./move file1.rtf file2.rtf CopyTest/
*/
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
```

```
#include <unistd.h>
#include <string.h>


#define MOVE_CMD "./move"
#define COPY_CMD "./copy"

//Prototype for copy function - If we're in MOVE mode, this will also unlink the original path.
int copy(int, char**);

//Size our mode string to the bigger of the two commands (always 7 in this case)
char mode[(sizeof(COPY_CMD) < sizeof(MOVE_CMD)) ?
                                sizeof(MOVE_CMD)     :     sizeof(COPY_CMD)];

//Feedback 2D arrays - Contains names of files to display whether they failed, and error
counts
char ** successfulActions;          int successes = 0;
char ** overwriteWarnings;         int warn_c_exist = 0;
char ** sourceErrors;                int warn_c_loc = 0;
char ** unlinkErrors;                 int warn_c_unlink = 0;

//Check the command used, verify the format, then try to execute command.
//parameters will be checked for proper inputs in the respective function
int main(int argc, char** argv)
{
      //Ensure we have the correct number of arguments
      if (argc < 3)
      {
            printf("Usage: /%s source1 [source2 ...] destination\n",
            //Uranary for command display
            (strcmp(mode, COPY_CMD) == 0) ? "copy" : "move");
            return EXIT_FAILURE;
      }

      //Save command type for future use (whether or not to unlink, and to give a synopsis).
```

```
strcpy(mode, argv[0]);

//Initialize user feedback variables
successfulActions = malloc(argc - 1);
overwriteWarnings = malloc(argc - 1);
sourceErrors = malloc(argc - 1);
unlinkErrors = malloc(argc - 1);

//Try to copy and send exit code if fatal error copying
if (EXIT_FAILURE == copy(argc, argv))
	return EXIT_FAILURE;



//Display a synopsis of our successes and failures while copying or moving
printf("\n---------- %s SYNOPSIS ----------\n\n",
(strcmp(mode, COPY_CMD) == 0) ? "COPY" : "MOVE");
//If there was at least one success
if (successes > 0)
{
	printf("The following %d %s successfully %s to %s\n",
	//Number of files copied or moved
	(argc - 2 - (warn_c_loc + warn_c_exist)),

	//Check how many files copied successfully
	((argc - 2 - (warn_c_loc + warn_c_exist)) > 1)?
	//Text format If multiple successes:
		"files were" :
	//Text format if one success
		"file was",

	//Text format for "copied" or "moved"
	(strcmp(mode, COPY_CMD) == 0) ? "copied" : "moved",

	//Text format for destination to display:
		argv[argc - 1]);
```

```
        //Display successful files
        for(int i = 0; i<successes; i++)
            printf(" -> %s\n",successfulActions[i]);
    }

    //Display errors, if any
    if (warn_c_exist > 0) {
        printf("WARNING: You chose not to overwrite the following %d %s:\n",
        warn_c_exist,
        (warn_c_exist > 1) ? "files" : "file");  //Grammar uranary

        //Print overwrite warnings
        for(int i = 0; i<warn_c_exist; i++)
            printf(" -> %s\n",overwriteWarnings[i]);

    }
    if (warn_c_loc > 0)
        printf("ERROR: %d %s the same source and destination:\n", warn_c_loc,
        (warn_c_loc > 1) ? "files have" : "file has");  //Grammar uranary

    //Were there any unlink errors?
    if (warn_c_unlink > 0)
        printf("WARNING: The following %d %s failed to delete!:\n", warn_c_unlink,
        (warn_c_unlink > 1) ? "files" : "file");  //Grammar

}

//All system logic occurs here - where the magic happens
//Copy file at path A to path B. If mode is MOVE, will attempt to unlink the original sources.
//We pass the arguments once again as parameters to iterate over the given sources.
int copy(int argc, char** argv)
{
    //The last argument is the destination
    char * dest = argv[argc - 1];
```

```
//Loop over each source
for (char **source = argv+1; *source != argv[argc - 1]; source++)
{
	//Hold the directory statistics
	struct stat path_stat;
   stat(dest, &path_stat);

  //If we're COPYING, check dest directory or device.
   if (strcmp(mode, COPY_CMD) == 0)
   { //If not a directory AND not a device (if all fail. we must be one!)
      if ((0 == S_ISDIR(path_stat.st_mode)) &&
          0 == S_ISCHR(path_stat.st_mode)  &&
          0 == S_ISBLK(path_stat.st_mode))
      {
            printf("Destination must be a device or directory!\n");
                  return EXIT_FAILURE;
      }
   }
   else  //If we're moving, destination must be a directory
      if (0 == S_ISDIR(path_stat.st_mode))
            {
                  printf("Destination must be a directory!\n");
                  return EXIT_FAILURE;
            }

      //Prepare source file name and path
      int count = 0;

      for(int i = strlen(*source) - 1; i>=0; i--)
      {//Read the source backwards for the file name

            if(*(*(source)+i) != '/')
                  {
                        count++;
```

```
                continue;
            }
            break;
}
//Make new string with enough room for the new file path.
char currentDest[count + strlen(dest)];

//currentDest has room to append the file name at the end. First we put the path.
strcpy(currentDest, dest);

//Add the file name at the end of the destination path
int end = strlen(dest); //This is where we will start adding new path characters.

char currentDestFileName[count + 1];        //Prepare to save file name
char currentSource[strlen(*source) - count]; //Prepare to chop off file name

//Append current file name to the end of our destination path.
//This will be the full path for the COPIED file, in the final location.
for (int i = 0; i <= count; i++)
{
    //Append the file name to the destination path
    currentDest[end+i] = *(*(source)+strlen(*source) - (count - i));

    //Save file name to display if we have an error
    currentDestFileName[i] = currentDest[end + i];
}

//Save the current source to check if it is equal to the destination, then throw
//an error as we were prohibited from copying a file to its own location as per spec
for (int i = 0; i< strlen(*source)-count; i++)
{
    //Store the source path to make sure we're not copying here
    currentSource[i] = *(*(source)+i);

    if (i == strlen(*source)-count - 1)
```

```
                currentSource[++i] = '\0';
    }

    //Increment warning count if trying to same-loc-copy
    if (0 == strcmp(currentSource, dest))
    {
        sourceErrors[warn_c_loc++] = *source;
        continue; //Do not copy this file, try the next one.
    }

    //Prepare to open the source file
    int fd;
    if (-1 == (fd = open(*source, O_RDONLY)))
    {
        printf("Source file not found!\n");
        return EXIT_FAILURE;
    }

    //make a buffer for the file contents
    char buf[BUFSIZ];
    int fileLength;
    if ((fileLength = read(fd, buf, BUFSIZ)) == -1)
    {
        printf("Failure reading the source file.\n");
        return EXIT_FAILURE;
    }

    //Define new file descriptor
    int new_fd;

    //Must check if a file here already exists. if so, ask to truncate
    if(access(currentDest, F_OK ) != -1)
{
    //File already exists here. Ask permission to overwrite
    char response[10];
```

```
    printf("\nCONFLICT: %s already exists here! Type 'yes' to overwrite:
    ",currentDestFileName);
    scanf("%s", response);

    //We did NOT give permission to overwrite - continue (keep looping sources)
    if(strcmp(response, "yes") != 0)
    {
        overwriteWarnings[warn_c_exist++] = *source;
        continue;
    }

}
//File has no conflicts, or we gave permission to OW.
//Therefore we can open the new descriptor, then write to it
new_fd = open(currentDest, O_WRONLY | O_TRUNC | O_CREAT, 0764);
        // 111 110 100 -> 0764 -> rwe / rw- / r--
//Try to write to the new file
if (-1 == write(new_fd, buf, fileLength))
    {
        printf("Failure copying the source file.\n");
        return EXIT_FAILURE;
    }
    //Here we succeeded if copying. If moving, still try to unlink source
    if (strcmp(mode, COPY_CMD) == 0)
        successfulActions[successes++] = *source;

    else //Try to unlink source path
    {
        if(-1 == unlink(*source))
        {
            unlinkErrors[warn_c_unlink++] = *source;
            continue; //Do not delete this file, try the next one.
        }
        else //we successfully moved the file!
            successfulActions[successes++] = *source;
```

```
            }

        }//For loop end brace - if we make it past here, there were no fatal errors.
        return EXIT_SUCCESS;
}
```