```c
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <dirent.h>
#include <libgen.h>

/*
 *ICSI333. System Fundamentals,
 *Spring 2022,
 *TA Sourav,
 *Kiran Aziz,
 *ID: 001440162
 */

/*
 *The copy function copies a file or a group of files to a specified path by using
 *system calls that read and write big blocks using the BUFSIZ macro. Additionally:
 *       -A file cannot be copied to itself.
 *       -If a file with such a name already exists in the destination folder,
 *        permission for overwriting is prompted.
 * @param source: String, or character array, for the name of the source file.
 * @param destination: String, or character array, for the name of the destination folder.
 */
void copy(char *source, char *destination){
 char buf[BUFSIZ]; //Character buffer to help with read and write operations.
 int sourceFD, destinationFD; //File descriptors for the source file and the destination folder.

 char *sourcePathCopy1 = strdup(source); //Duplicates path to source file to extract directory name.
 char *dir = dirname(sourcePathCopy1);
 char *sourcePathCopy2 = strdup(source); //Duplicates path to source file to extract file name.
 char *filename = basename(sourcePathCopy2);

 //Creates new path based on path to destination folder.
 char fullDestination[strlen(destination) + 20];
 strcpy(fullDestination, destination);
 fflush(stdout);
 strcat(fullDestination, "/");
 fflush(stdout);
 strcat(fullDestination, filename);

 //If the source file path is identical to the destination path,
 //then the paths are the same and a file cannot copy to itself.
 if(strcmp(source, fullDestination) == 0){
  fprintf(stderr, "Error: A file should not be copied to itself.\n");
  return;
 }

 //Describes success or failure for system call open()
 //Returns -1 if there is an error and prints an error message.
 sourceFD = open(source, O_RDONLY);
 if (sourceFD < 0) {
  fprintf(stderr, "Error: Cannot open or find source file.\n");
  perror("open");
  close(sourceFD);
  return;
 }

 //Checks if the source file is a regular file using system call stat().
 //If the source file is not a regular file, then an error
 //message is printed and the program is terminated.
 struct stat sb;
 if(stat(source, &sb) != -1){
  if(S_ISREG(sb.st_mode)){
```

```c
  fprintf(stderr, "Source file is a regular file.\n");
 }
 else{
  fprintf(stderr, "Error: The source file must be a regular file.\n");
  exit(1);
 }
}

//Checks if the destination folder is a directory or a device using system call stat().
//If the destination folder is neither a directory or a device,
//then an error message is printed and the program is terminated.
struct stat s;
if(stat(destination, &s) != -1){
 if(S_ISDIR(s.st_mode)){
  fprintf(stderr, "Destination is a directory.\n");
 }
 else if(S_ISCHR(s.st_mode)){
  fprintf(stderr, "Destination is a character device.\n");
 }
 else if(S_ISBLK(s.st_mode)){
  fprintf(stderr, "Destination is a block device.\n");
 }
 else{
  fprintf(stderr, "Error: The destination must be a directory or a device.\n");
  close(sourceFD);
  exit(1);
 }
}

DIR *folder; //Creates a pointer to the directory/destination folder.
folder = opendir(destination);
struct dirent *dirFile; //Creates a pointer to a file in the directory.

//Folder will return NULL if there is no directory found or there is an error.
//Prints out error message and terminates program.
if(folder == NULL){
 fprintf(stderr, "Error: Unable to read directory.\n");
 close(sourceFD);
 closedir(folder);
 exit(1);
}

//If pointer named folder returns a directory, then the directory will be traversed.
while(dirFile = readdir(folder)){
 //If a file in the directory is the same name
 //as the source file name, then the user is prompted
 //to either allow or deny permission for overwriting.
 if(strcmp(dirFile->d_name, filename) == 0){
  char ch; //Character that holds the user's input to allow (Y) or deny (N) permission to overwrite.

  //Prints warning message to user.
  printf("Warning: File with such a name already exists in the destination folder, permission to overwrite? Enter Y or N: ");
  fflush(stdout);
  scanf(" %c", &ch); //User input.


  //If user denies permission, then this source file is not copied.
  //If there is more than one source file to be copied, then the program
  //continues onto the next source file.
  if(ch == 'N' || ch == 'n'){
   fprintf(stderr, "No overwriting.\n");
   close(sourceFD);
   closedir(folder);
   return;
  }else{
   break;
```

```
    }

   }

  }


  //Closes directory after traversal.
  closedir(folder);

  //Describes success or failure for system call open().
  //Creation Flags:
  //O_WRONLY - If destination file exists, buffer will write to it.
  //O_CREAT - If destination file doesn't exist, it is first created and then written to.
  //O_TRUNC - If overwriting is allowed, the existing file in the directory has its data
  //          removed and replaced by that of the source file.
  //Open Modes:
  //S_IRUSR - File can be read by user.
  //S_IWUSR - File can be written by user.
  //S_IRGRP - File can be read by users in group.
  //S_IROTH - File can be read by others.
  destinationFD = open(fullDestination, O_WRONLY | O_CREAT | O_TRUNC, S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH);

  //Reads data from source file to the buffer.
  //Using BUFSIZ macro to determine the size of the buffer.
  while(read(sourceFD, &buf, (size_t) BUFSIZ)) {
   //If system call read() fails, then an error message is shown.
   //Moves onto next source file if there is more than one source file.
   if(read(sourceFD, &buf, (size_t) BUFSIZ) == -1){
    fprintf(stderr, "Error in read.\n");
    close(sourceFD);
    close(destinationFD);
    return;
   }

   //Writes data from the buffer to the destination file.
   write(destinationFD, &buf, (size_t) BUFSIZ);

   //If system call write() fails, then an error message is shown.
   //Moves onto next source file if there is more than one source file.
   if(write(destinationFD, &buf, (size_t) BUFSIZ) == -1){
    fprintf(stderr, "Error in write.\n");
    close(sourceFD);
    close(destinationFD);
    return;
   }
  }

  //Outputs the action after the copying operation is finished.
  printf("%s successfully copied to %s.\n", source, destination);
  fflush(stdout);

  //Closes source file descriptor and destination file descriptor.
  close(sourceFD);
  close(destinationFD);

 }


/*
 *The move function moves a file or a group of files to a specified path.  When performing
 *the move, it copies a file to the new location then deletes the file (unlinks the old path).
 *Physical copying of all bytes takes time; the more efficient way, as seen below, is
 *by creating a new link (the new path linked to the old bytes on a disk).
 *If the action cannot be performed through linking because of the OS, then the file
 *is copied and the source is deleted.
 * @param source: String, or character array, for the name of the source file.
```

```c
 * @param destination: String, or character array, for the name of the destination folder.
 */
void move(char *source, char *destination){
 char *sourcePathCopy1 = strdup(source); //Duplicates path to source file to extract directory name.
 char *dir = dirname(sourcePathCopy1);
 char *sourcePathCopy2 = strdup(source); //Duplicates path to source file to extract file name.
 char *filename = basename(sourcePathCopy2);

 //Creates new path based on path to destination folder.
 char fullDestination[strlen(destination) + 20];
 strcpy(fullDestination, destination);
   fflush(stdout);

 strcat(fullDestination, "/");
 fflush(stdout);
 strcat(fullDestination, filename);

 //Checks if the source file is a regular file using system call stat().
 //If the source file is not a regular file, then an error
 //message is printed and the program is terminated.
 struct stat sb;
 if(stat(source, &sb) != -1){
  if(S_ISREG(sb.st_mode)){
   fprintf(stderr, "Source file is a regular file.\n");
  }
  else{
   fprintf(stderr, "Error: The source file must be a regular file.\n");
   exit(1);
  }
 }

 //Checks if the destination folder is a directory using system call stat().
 //If the destination folder is not a directory,
 //then an error message is printed and the program is terminated.
 struct stat s;
 if(stat(destination, &s) != -1){
  if(S_ISDIR(s.st_mode)){
   fprintf(stderr, "Destination is a directory.\n");
  }
  else{
   fprintf(stderr, "Error: The destination must be a directory.\n");
   exit(1);
  }
 }

 //Creates a hard link from the source file to the destination file.
 int l = link(source, fullDestination);

 //If linking fails, then an error message is printed
 //Moves onto next source file if there is more than one source file.
 if(l < 0){
  fprintf(stderr, "Error: Can't link to directory %s.\n", destination);
  perror("link");
  return;

 }

 //Unlinks source file, which deletes the source file.
 int u = unlink(source);

 //If unlinking fails, then an error message is printed
 //and the program is terminated.
 if(u < 0){
  fprintf(stderr, "Error: Can't remove source file.\n");
  exit(1);
 }
 //Outputs the action after the moving operation is finished.
 printf("%s successfully moved to %s.\n", source, destination);
 fflush(stdout);
}

/*
 *The main function performs the system actions - copying or moving a file or a group
 *of files to a specified path.  The program acts differently depending on the usage
 * It performs copying if used as:
 *           copy source1 [source2 ...] destination
 * It performs moving if used as:
 *           move source1 [source2 ...] destination
 *
 * After the program has decided on the required action (copying or moving), it checks
 * the validity of the destination.
 * Then, the program processes each source file to copy or move. If a source file does not
 * exist, the error message is generated, and the next file must be tried.
 * @param argc: Integer holding the number of command line arguments.
 * @param argv: Array of string values, with each value being an command line argument
 *             that is passed onto the main function.
 */
void main(int argc, char *argv[]){
 int numOfSources = argc - 2; //Integer holding the number of source files in command
        //command line arguments array.
 //If user's command is copy, then it copies all source files
 //to the desired destination.
 if(strcmp("./copy", argv[0]) == 0){
  int i = 1; //Integer holding the value for the starting index of source files.

  //Processes each source file from command line.
  while(i <= numOfSources){
   copy(argv[i], argv[argc - 1]);
   i++;
  }
 }


 //If user's command is move, then it moves all source files
 //to the desired destination.
 else if(strcmp("./move", argv[0]) == 0){
  int i = 1; //Integer holding the value for the starting index of source files.

  //Processes each source file from command line.
  while(i <= numOfSources){
   move(argv[i], argv[argc-1]);
       i++;
  }
 }
}
```