```c
/*John Brady & Albert Owusu-Asare
Boxes 3119 & 4497
December 8, 2014
CSC213, Weinman
Lab 10: Filesystems
mycp.c - error checking and reporting version of cp
*/
/* File copy program.  */
/* From Andrew S. Tanenbaum, _Modern_Operating_Systems 3/e, p. 266.
 * Transcribed by Janet Davis, October 8, 2010.
 *Modified by Albert & John for Jerod Weinman's Lab 10
 */

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

int main(int argc, char *argv[]);   /* ANSI prototype */

#define BUF_SIZE 4096           /* use a buffer size of 4096 bytes */
#define OUTPUT_MODE 0700        /* protection bits for output file */

int main(int argc, char *argv[])
{
    int in_fd, out_fd, rd_count, wt_count;
    char buffer[BUF_SIZE];

    if (argc != 3)
    {
        fprintf(stderr, "Syntax: mycp <source> <destination>\n");
        exit(EXIT_FAILURE);        /* syntax error if argc is not 3 */
    }

    /* Open the input file and create the output file */
    in_fd = open(argv[1], O_RDONLY);             /* open the source file */

    if (in_fd < 0)
    {
        perror("Open Error");
        exit(EXIT_FAILURE);         /* if it cannot be opened, exit */
    }
    //Copy Permissions to struct
    struct stat orig_stats;
    fstat(in_fd, &orig_stats);

    out_fd = creat(argv[2], orig_stats.st_mode ); //Create  destination file
    //based on testing, EISDIR should be 21
    if (errno == EISDIR)
    {
        char *orig_name = basename(argv[1]); //gives us source name
        char *dest_path = malloc(sizeof(argv[1]) +
                                 sizeof(argv[2] + 3)); //declares destination
        strcpy(dest_path, argv[2]); //add directory name
        strcat(dest_path, "/"); //add slash into directory
        strcat(dest_path, orig_name); //add output file name
        out_fd = creat(dest_path, orig_stats.st_mode); //retry copy


    }
    if (out_fd < 0)
    {
        perror("Create Error");
        exit(EXIT_FAILURE);        /* if it cannot be created, exit */
    }
    /* Copy loop */
    while (1)
    {
        rd_count = read(in_fd, buffer, BUF_SIZE); /* read a block of data */
        if (rd_count <= 0)
        {
            //if rd_count other than EOF, err
            if (rd_count < 0) perror("Read Error");
            break;      /* if end of file or error, exit loop */
        }
        wt_count = write(out_fd, buffer, rd_count);         /* write data */
        if (wt_count <= 0)
        {
            perror("Write Error");
            exit(EXIT_FAILURE);           /* wt_count <= 0 is an error */
        }
    }



    /* Close the files */
    close(in_fd);
    close(out_fd);
    if (rd_count == 0)               /* no error on last read */
        exit(EXIT_SUCCESS);
    else                                /* error on last read */
    {
        perror("Close Error");
        exit(EXIT_FAILURE);
    }
}
```

```c
/*John Brady & Albert Owusu-Asare
Boxes 3119 & 4497
December 8, 2014
CSC213, Weinman
Lab 10: Filesystems
myls.c: prints information about the current or listed directory
NOTE: Skeleton (with includes and ANSI prototype) copied from myrm.c
*/

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


int main(int argc, char *argv[]);   // ANSI prototype */


int main(int argc, char *argv[])
{


    if (argc > 2)
    {
        fprintf(stderr, "Syntax: myls <directory>/<>\n");
        exit(EXIT_FAILURE);    // syntax error if argc is more than 2 */
    }

    //now directory is the directory we want to print. pass to opendir

    DIR *directory;
    struct dirent *ent;
    //if we have an argument, open the listed directory
    if (argc == 2)
        directory = opendir(argv[1]);
    else
        //otherwise open the current directory
        directory = opendir(".");
    //if nothing in directory
    if (directory == NULL)
    {
        perror("Open Error");
        exit(EXIT_FAILURE);         // if it cannot be opened, exit
    }
    else
    {
        //iterate through directory
        while ((ent = readdir(directory)) != NULL)
        {
            //if the file is not hidden (marked by a leading .)
            if ('.' != ent->d_name[0])
            {
                struct stat buf;
                int error;
                //create path; argv's length plus max d_name size

                char *filepath = malloc(sizeof(argv[1]) + 255 * sizeof(char));

                if (argc == 2)
                {
                    strcpy(filepath, argv[1]);
                }
                else
                {
                    strcpy(filepath, ".");
                }
                strcat(filepath, "/");
                strcat(filepath, ent->d_name);
                //store statistics in buf
                error = stat(filepath, &buf);
                if (error < 0)
                {
                    perror("Stat");
                    exit(EXIT_FAILURE); // if it cannot be opened, exit */
                }

                printf("%s\t%d\n", ent->d_name, buf.st_size);
            }

        }//while
        //close directory
        closedir(directory);

    }//else
    exit(EXIT_SUCCESS);

}
```

```c
/*John Brady & Albert Owusu-Asare
Boxes 3119 & 4497
December 8, 2014
CSC213, Weinman
Lab 10: Filesystems
mymv.c: moves/renames a file
NOTE: Skeleton (with includes and ANSI prototype) copied from myrm.c
*/

#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
#include <string.h>
#include <libgen.h>
#include <errno.h>
#include <utime.h>


int main(int argc, char *argv[]);   // ANSI prototype */


int main(int argc, char *argv[])
{
    //check for correct number of arguments
    if (argc != 3)
    {
        fprintf(stderr, "Syntax: mymv <oldpath> <newpath>\n");
        exit(EXIT_FAILURE);        /* syntax error if argc is not 3*/
    }

    int error;
    error = rename(argv[1], argv[2]);
    //if our source file was a directory
    if (errno == EISDIR)
    {
        //allocate for an array to store the splits
        const char slash[2] = "/";
        char *token;

        char *input = malloc(sizeof(argv[1]));
        char *output = malloc(sizeof(argv[1]));
        strcpy(input, argv[1]);

        token = strtok(input, slash);

        while (token != NULL)
        {
            strcpy(output, token);
            token = strtok(NULL, slash);

        }

        char *updated_path = malloc(sizeof(argv[1]) + sizeof(argv[2]));
        strcpy(updated_path, argv[2]);
        strcat(updated_path, "/");
        strcat(updated_path, output);

        error = rename(argv[1], updated_path);

    }

    if (error < 0)
    {
        perror("rename Error");
        exit(EXIT_FAILURE);
    }
}
```

```c
/*John Brady & Albert Owusu-Asare
Boxes 3119 & 4497
December 8, 2014
CSC213, Weinman
Lab 10: Filesystems
mycp.c - error checking and reporting version of cp
NOTE: adapted from mycp.c to use the includes, ANSI thing, and open/close
*/

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


int main(int argc, char *argv[]);   /* ANSI prototype */


int main(int argc, char *argv[])
{
    int in_fd;

    if (argc != 2)
    {
        fprintf(stderr, "Syntax: myrm <filename>\n");
        exit(EXIT_FAILURE);        /* syntax error if argc is not 2 */
    }

    /* Open the input file  */
    in_fd = open(argv[1], O_RDONLY);             // open the source file
    struct stat stats;
    fstat(in_fd, &stats);

    if (S_ISDIR(stats.st_mode))
    {
        fprintf(stderr, "%s refers to a directory.\n", argv[1]);
        exit(EXIT_FAILURE);
    }
    if (in_fd < 0)
    {
        perror("Open Error");
        exit(EXIT_FAILURE);         /* if it cannot be opened, exit */
    }

    /* Close the file */
    close(in_fd);

    //delete the file
    unlink(argv[1]);

}
```

```c
/*John Brady & Albert Owusu-Asare
Boxes 3119 & 4497
December 8, 2014
CSC213, Weinman
Lab 10: Filesystems
mytouch.c: updates date modified time to current system time
NOTE: Skeleton (with includes and ANSI prototype) copied from myrm.c
*/

#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
#include <string.h>
#include <libgen.h>
#include <errno.h>
#include <utime.h>


#define OUTPUT_MODE 0644

int main(int argc, char *argv[]);   /* ANSI prototype */


int main(int argc, char *argv[])
{
    int out_fd;

    if (argc != 2)
    {
        fprintf(stderr, "Syntax: mytouch <filename>\n");
        exit(EXIT_FAILURE);        /* syntax error if argc is not 2 */
    }
    int time_check;
    time_check = utime(argv[1], NULL);
    if (time_check < 0)
    {
        if (errno == ENOENT)
        {
            //create a file with system default permissions
            out_fd = creat(argv[1], OUTPUT_MODE ); //Create destination file
        }
        else
        {
            perror("Utime Error");
            exit(EXIT_FAILURE);
        }       /* if it cannot be created, exit */
    }
    //if utime returns that a file does not exist

    if (out_fd < 0)
    {
        perror("Create Error");
        exit(EXIT_FAILURE);        /* if it cannot be created, exit */
    }
    /* Close the file */
    close(out_fd);

}
```

```
#John Brady & Albert Owusu-Asare
#Boxes 3119 & 4497
#December 8, 2014
#CSC213, Weinman
#Lab 10: Filesystems
# This document contains compile commands for the makefile

CC = gcc
CCOPTS = -c -g -Wall -ggdb
LINKOPTS = -g
PROGRAMS = mycp myrm mytouch mymv myls

all: $(PROGRAMS)

# Implicit rule: Make program named x from x.c
% :: %.c
        $(CC) $(LINKOPTS) -o $@ $^

clean:
        rm -rf *.o *~ *.err *.out $(PROGRAMS)
```

John Brady & Albert Owusu-Asare
Boxes 3119 & 4497
December 8, 2014
CSC213, Weinman
Lab 10: Filesystems
README: This file is a readme for the utilities in this folder, as well as for the lab. Descriptions, statements of correctness, citations

Descriptions:

mycp <source> <destination> : copies a file from its source to a destination location, whether it be a new file of name <destination> (or <directory/destination>) or just a directory named <destination>. It will exit and return an EXIT_FAILURE status if there is an error, otherwise will execute and close, returning EXIT_SUCCESS. File permissions also transfer with the file data.

myrm <filename> : removes the non-directory file types specified in the arguments specified on the command line. Will report an error if the file does not exist, path is a directory, or if the syntax is incorrect.

mytouch <filename> : updates a pre-existing file (or directory) to have its modification time match the current system time. If no such file exists, a new file will be created with the modification time as the current system time. Will exit and return an error status if there is a wrong number of arguments, utime call fails, or creat call fails.

mymv <oldpath> <newpath> : moves a file named <oldpath> to a directory/file named <newpath>

myls <directory>/<> : lists all files and sizes in the current directory or the specified directory. Will present an error if there is an error fetching file size statistics, or if the syntax isn't correct.

NOTE on mymv: We will consider advantages and disadvantages to the three approaches listed in Lab 10.

For the first approach, all hard links to the original file would be lost (from unlink(2)) as the original file is deleted and a copy of the file is made in the new location. This would be able to traverse filesystems, but since the data is being copied the performance would be less than in other approaches and there would be a higher risk of data loss during the copying procedure in the event of a system crash.

For the second approach, this would alleviate the performance issues of the first approach because we only point to the original data using a hard link instead of copying the file over. Moreover, unlinking the source pathname would not affect any of the other possible hard links pointing to it, since we just created a second hard link which by **unlink**(2) will not delete the file contents. However, a disadvantage would be that a hard link cannot exist in one file system and point to data on another file system, so this feature would be unavailable.

For the third approach, we follow a similar method to the second approach, but with additional mechanisms in place. The rename(2) man page suggests that it only moves a file if it is required, but reading more suggests that the command does not work across file systems, which is a big disadvantage. Rename also has the capability to do nothing if the new path is already a hard link pointing to the same file, which should increase performance over deleting and remaking a hard link. Again, we see that rename does not work across filesystems, so we would be limited to moving a file within a filesystem. Furthermore, mv has the feature that you don't need to name the destination path file; you can just name the directory it will reside in and the filename will be automatically copied. Rename does not support this, but we have added a mechanism that allows for this behavior to work.

Statements of Correctness:

Note: For adding in perror output for issues with mycp's execution, we are unable to test whether or not our errors work correctly, since producing a situation where they would be called is very difficult. However, we can test (and print) for the wrong number of arguments or a missing source file, and we also test copying a file to a directory.

For myrm, we test our program with the wrong number of arguments and with no arguments, which prints out the syntax. We also test it with a non-existent file, which prints an open error. We also successfully test myrm with a regular file, a file in a subdirectory, and a directory, which prints a directory error.

For mytouch, we test it with a regular file, and then test it with the wrong syntax, which results in the printout of a proper syntax. We then remove a file to ensure it does not exist, and then call mytouch on that filename, which creates the file and sets its modified time to the current time of calling.

For mymv, we test our program with the wrong number of arguments and with only one argument, which prints out the syntax. We also test a non-existent source file, which presents an appropriate error. We then check functionality by moving a file from the current directory to a subdirectory, and moving a file from a subdirectory to the current directory. Finally, we try moving a file to a new directory, but the file name is unspecified. The program fills in the blank file name with the source file name, just as mv does.

For myls, we test our program with the wrong number of arguments, and we see proper syntax printed. We then test it with the current directory and see that printed. We then test it with no arguments, which still prints the current directory, as it should. Then we make a subdirectory, fill it with two files, and list the contents of that subdirectory.

Citations:
Much assistance was provided by the man documentation for the various calls. We also talked with Jerod Weinman regarding the choice of implementation for mymv.c.

```
boole$ ./batch
++ make clean mycp myrm mytouch mymv myls
rm -rf *.o *~ *.err *.out mycp myrm mytouch mymv myls
gcc -g -o mycp mycp.c
gcc -g -o myrm myrm.c
gcc -g -o mytouch mytouch.c
gcc -g -o mymv mymv.c
gcc -g -o myls myls.c
++ touch test
++ mkdir di
mkdir: cannot create directory ‘di’: File exists
++ r
./batch: line 12: r: command not found
++ mycp test test2
++ mycp blah blah blah
Syntax: mycp <source> <destination>
++ mycp blah test
Open Error: No such file or directory
++ mycp test dir
++ myrm test1 test2
Syntax: myrm <filename>
++ myrm testtttt
Open Error: Bad file descriptor
++ myrm
Syntax: myrm <filename>
++ myrm test2
++ myrm ./dir/test
Open Error: Bad file descriptor
++ myrm dir
++ mytouch test
++ mytouch test test test
Syntax: mytouch <filename>
++ rm test3
rm: cannot remove ‘test3’: No such file or directory
++ mytouch test3
++ mymv x y z w
Syntax: mymv <oldpath> <newpath>
++ mymv x
Syntax: mymv <oldpath> <newpath>
++ mymv foo bar
rename Error: No such file or directory
++ mytouch test3
++ mymv test3 dir/test3
rename Error: No such file or directory
++ mytouch dir/test4
Create Error: No such file or directory
++ mymv dir/test4 test4
rename Error: No such file or directory
++ mymv test4 dir/test4
rename Error: No such file or directory
++ mkdir testingdir
++ mymv dir/test4 testingdir
rename Error: No such file or directory
++ myls x y z
Syntax: myls <directory>/<>
++ myls .
test    0
Makefile        409
myls.c  2401
testingdir      4096
lsdir   4096
test3   0
di      4096
mycp    13385
batch   1443
myls    12315
README  5534
mytouch 10484
source.pdf      18521
mycp.c  2979
mymv.c  1574
mymv    11124
myrm    11412
myrm.c  1201
source.pdf.pdf  2217
mytouch.c       1393
++ myls
test    0
Makefile        409
myls.c  2401
testingdir      4096
lsdir   4096
test3   0
di      4096
mycp    13385
batch   1443
myls    12315
README  5534
mytouch 10484
source.pdf      18521
mycp.c  2979
mymv.c  1574
mymv    11124
myrm    11412
myrm.c  1201
source.pdf.pdf  2217
mytouch.c       1393
++ mkdir lsdir
mkdir: cannot create directory ‘lsdir’: File exists
++ touch lsdir/1
++ touch lsdir/2
++ myls lsdir
2       15
1       0
++ rm -r test test3 testingdir dir
rm: cannot remove ‘dir’: No such file or directory
boole$
```

```
#John Brady and Alber Owusu-Asare
#Boxes 3119 and 4497
#batch: a batch file to compile and test for correctness.
#!/bin/bash
set -x

make clean mycp myrm mytouch mymv myls

#initialize files
touch test
mkdir di
r
mycp test test2 #test copy
mycp blah blah blah #test num arguments
mycp blah test #test no input file error
mycp test dir #test copy to a directory

myrm test1 test2 #test wrong number of arguments
myrm testtttt #test with nonexistent file
myrm #test with no arguments
myrm test2 #remove test2 file
myrm ./dir/test #remove file in subdirectory
myrm dir #remove directory

mytouch test #update time of pre-existing file
mytouch test test test # wrong number of arguments
rm test3 #ensure file does not exist
mytouch test3 #touch a non-existing file

mymv x y z w #wrong number of arguments
mymv x # wrong number of arguments
mymv foo bar #test nonexistent files
mytouch test3 # create file in cur directory
mymv test3 dir/test3 #move a file to a subdirectory
mytouch dir/test4 #create a file inside a directory
mymv dir/test4 test4 #move file from subdir to cur dir
mymv test4 dir/test4 #set back in place
mkdir testingdir
mymv dir/test4 testingdir #move to a dir, no name specified

myls x y z # wrong arguments
myls . #specific current directory
myls #unspecific current directory
mkdir lsdir #make directory
touch lsdir/1 #make file
touch lsdir/2 #make file
myls lsdir #list subdirectory

#clean up files
rm -r test* dir
```