// Camille Chow

// ECE357 Assignment 2

#include <sys/stat.h>

#include <dirent.h>

#include <stdio.h>

#include <errno.h>

#include <string.h>

#include <unistd.h>

#include <fcntl.h>

#include <limits.h>

//Global Variables

ino\_t target\_ino;

off\_t target\_size;

dev\_t target\_dev;

char\* target\_name;

const int buff\_size = BUFSIZ;

//Function to compare contents of files

//returns 1 if identical, returns 0 if not identical, returns -1 on error

int compareFiles(char\* path) {

int fd1, fd2;

char\* b1[buff\_size], b2[buff\_size];

//open files

if ((fd1 = open(target\_name, O\_RDONLY)) < 0) {

fprintf(stderr, "Warning: Can't open target file '%s' for reading: %s\nCan't check for duplicate\n", target\_name, strerror(errno));

return -1;

}

if ((fd2 = open(path, O\_RDONLY)) < 0) {

fprintf(stderr, "Warning: Can't open file '%s' for reading: %s\nCan't check if duplicate\n", path, strerror(errno));

return -1;

}

int n,m;

//read to buffer

while ((n = read(fd1, b1, buff\_size)) != 0 && (m = read(fd2, b2, buff\_size)) != 0) {

if (n < 0) {

fprintf(stderr, "Error reading from target file '%s': %s\nCan't check for duplicate\n", target\_name, strerror(errno));

return -1;

}

else if (m < 0) {

fprintf(stderr, "Error reading from target file '%s': %s\nCan't check if duplicate\n", path, strerror(errno));

return -1;

}

else if (memcmp(b1,b2,n) != 0) {

return 0;

}

}

return 1;

}

//Recursive searching function

void searchFiles(char \*directory, int canTraverse) {

DIR \*dir;

struct dirent \*entry;

if (!(dir = opendir(directory))) {

fprintf(stderr, "Warning: Could not open directory %s: %s\n", directory, strerror(errno));

return;

}

while ((entry = readdir(dir)) != NULL) {

//current path

char path[PATH\_MAX];

sprintf(path, "%s/%s", directory, entry->d\_name);

//run stat on entry

struct stat st;

if (stat(path,&st) < 0) {

fprintf(stderr, "Warning: Could not run stat on entry %s: %s\n", path, strerror(errno));

continue;

}

mode\_t mode = st.st\_mode;

ino\_t ino = st.st\_ino;

off\_t size = st.st\_size;

dev\_t dev = st.st\_dev;

//if entry is another directory, recursively search

if ((mode & S\_IFMT) == S\_IFDIR) {

if (strcmp(entry->d\_name, ".") == 0 || strcmp(entry->d\_name, "..") == 0) {

continue;

}

searchFiles(path, ((mode & S\_IXOTH) == S\_IXOTH) && canTraverse);

}

//symlink handling

else if ((mode & S\_IFMT) == S\_IFLNK) {

//find contents of link

char link[PATH\_MAX];

if (readlink(path,link,sizeof(link)) < 0) {

fprintf(stderr, "Warning: Could check contents of symlink %s: %s\n", path, strerror(errno));

continue;

}

//run stat on link

struct stat st2;

if (stat(link,&st2) < 0) {

fprintf(stderr, "Warning: Could not run stat on contents of symlink %s (%s): %s\n", path, link, strerror(errno));

continue;

}

mode\_t mode = st2.st\_mode;

ino\_t ino = st2.st\_ino;

off\_t size = st2.st\_size;

dev\_t dev = st2.st\_dev;

if ((mode & S\_IFMT) == S\_IFREG) {

//resolves to target

if (ino == target\_ino && dev == target\_dev) {

printf("%s\tSYMLINK RESOLVES TO TARGET\n",path);

}

//resolves to duplicate

else if (size == target\_size && compareFiles(link) == 1) {

printf("%s\tSYMLINK (%s) RESOLVES TO DUPLICATE\n",path,link);

}

}

else {

printf("%s links to something not a file, skipping\n", path);

}

}

//regular file handling

else if ((mode & S\_IFMT) == S\_IFREG) {

char\* perm\_string;

if ((mode & S\_IROTH) != 0 && canTraverse) {

perm\_string = "OK READ by OTHER";

}

else {

perm\_string = "NOT READABLE by OTHER";

}

//check for hardlink

if (ino == target\_ino && dev == target\_dev) {

printf("%s\tHARD LINK TO TARGET\t%s\n",path,perm\_string);

}

//check for duplicate

else if (size == target\_size && compareFiles(path) == 1) {

nlink\_t links = st.st\_nlink;

printf("%s\tDUPLICATE OF TARGET (nlink=%d)\t%s\n",path,links,perm\_string);

}

}

//other file type

else {

printf("%s not a directory, regular file, or symlink, skipping\n", path);

}

}

closedir(dir);

}

//main function

int main(int argc, char\*\*argv) {

if (argc != 3) {

fprintf(stderr, "Incorrect number of input arguments\n");

return -1;

}

target\_name = argv[1];

char\* starting\_directory = argv[2];

DIR\* dir;

struct stat st, st\_dir;

int t;

if (stat(target\_name,&st) < 0) {

fprintf(stderr, "FATAL ERROR: Could not run stat on target file '%s': %s\n", target\_name, strerror(errno));

return -1;

}

if (!(dir = opendir(starting\_directory))) {

fprintf(stderr, "FATAL ERROR: Could not open directory '%s': %s\n", starting\_directory, strerror(errno));

return -1;

}

if (stat(starting\_directory,&st\_dir) < 0) {

fprintf(stderr, "Warning: Could not run stat on directory '%s': %s\nCan't determine traversal permissions\n", starting\_directory, strerror(errno));

t = 0; //assume others can't traverse

}

else {

mode\_t mode = st\_dir.st\_mode;

t = (mode & S\_IXOTH) == S\_IXOTH; //traversal permissions of starting directory

}

target\_ino = st.st\_ino;

target\_size = st.st\_size;

target\_dev = st.st\_dev;

searchFiles(starting\_directory,t);

return 0;

}