#include <stdio.h>

int main() {

FILE \*source\_file = fopen("/path/to/source/file.txt", "rb");

FILE \*dest\_file = fopen("/path/to/destination/file.txt", "wb");

// Check if the files were opened successfully

if (source\_file == NULL || dest\_file == NULL) {

printf("Failed to open files.\n");

return 1;

}

// Transfer the contents of the source file to the destination file

while (!feof(source\_file)) {

char buffer[1024];

size\_t bytes\_read = fread(buffer, 1, sizeof(buffer), source\_file);

fwrite(buffer, 1, bytes\_read, dest\_file);

}

// Check if the transfer is completed

long source\_position = ftell(source\_file);

long dest\_position = ftell(dest\_file);

if (source\_position == dest\_position) {

printf("File transfer completed.\n");

} else {

printf("File transfer failed.\n");

}

// Close the files

fclose(source\_file);

fclose(dest\_file);

return 0;

}

/////////////////////

#include <stdio.h>

#include <dirent.h>

int main() {

DIR \*dir = opendir("directory\_name");

if (dir == NULL) {

perror("Error opening directory");

return 1;

}

struct dirent \*entry;

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

// Skip special directories

if (entry->d\_type == DT\_DIR || entry->d\_name[0] == '.') {

continue;

}

// Build the source and destination file paths

char source\_path[256];

char dest\_path[256];

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

sprintf(dest\_path, "destination\_directory/%s", entry->d\_name);

// Open the source and destination files

FILE \*source\_file = fopen(source\_path, "rb");

FILE \*dest\_file = fopen(dest\_path, "wb");

// Transfer the contents of the source file to the destination file

while (!feof(source\_file)) {

char buffer[1024];

size\_t bytes\_read = fread(buffer, 1, sizeof(buffer), source\_file);

fwrite(buffer, 1, bytes\_read, dest\_file);

}

// Check if the transfer is completed

long source\_position = ftell(source\_file);

long dest\_position = ftell(dest\_file);

if (source\_position == dest\_position) {

printf("File %s transfer completed.\n", entry->d\_name);

} else {

printf("File %s transfer failed.\n", entry->d\_name);

}

// Close the files

fclose(source\_file);

fclose(dest\_file);

}

closedir(dir);

return 0;

}

/////////////////////// file status if it open //////////////

#include <stdio.h>

int main() {

FILE \*file = fopen("file\_name", "ab");

if (file == NULL) {

perror("Error opening file");

return 1;

}

// Write some data to the file

char data[1024];

fwrite(data, sizeof(char), sizeof(data), file);

// Check if the file is open for writing

if (ftell(file) == -1L) {

printf("File is not open for writing.\n");

} else {

printf("File is open for writing.\n");

// Check if all contents have been written until the last block

if (fflush(file) != 0) {

printf("Error flushing file.\n");

} else if (fseek(file, 0L, SEEK\_END) != 0) {

printf("Error seeking to end of file.\n");

} else {

long last\_block\_size = ftell(file) % 512;

if (last\_block\_size != 0) {

printf("File has not been written until the last block.\n");

} else {

printf("File has been written until the last block.\n");

}

}

}

fclose(file);

return 0;

}

///////////

#include <stdio.h>

#include <stdlib.h>

int main() {

char \*filename = "file\_name";

char \*directory = "/path/to/directory";

// Write the file to the server's file system

FILE \*file = fopen(filename, "wb");

if (file == NULL) {

perror("Error opening file");

return 1;

}

fwrite(file\_data, sizeof(char), file\_size, file);

fclose(file);

// Check if the file is open for writing until the last block

file = fopen(filename, "ab");

if (file == NULL) {

perror("Error opening file");

return 1;

}

if (ftell(file) == -1L) {

printf("File is not open for writing.\n");

} else {

printf("File is open for writing.\n");

// Check if all contents have been written until the last block

if (fflush(file) != 0) {

printf("Error flushing file.\n");

} else if (fseek(file, 0L, SEEK\_END) != 0) {

printf("Error seeking to end of file.\n");

} else {

long last\_block\_size = ftell(file) % 512;

if (last\_block\_size != 0) {

printf("File has not been written until the last block.\n");

} else {

printf("File has been written until the last block.\n");

}

}

}

fclose(file);

// Move the file to the specified directory

char \*move\_command = malloc(strlen(filename) + strlen(directory) + 16);

sprintf(move\_command, "mv %s %s", filename, directory);

system(move\_command);

free(move\_command);

return 0;

}

/////////////////////

#include <stdio.h>

#include <stdlib.h>

int main() {

char \*filename = "file\_name";

char \*directory = "/path/to/directory";

// Write the file to the server's file system

FILE \*file = fopen(filename, "wb");

if (file == NULL) {

perror("Error opening file");

return 1;

}

fwrite(file\_data, sizeof(char), file\_size, file);

fclose(file);

// Check if the file is open for writing until the last block

file = fopen(filename, "ab");

if (file == NULL) {

perror("Error opening file");

return 1;

}

if (ftell(file) == -1L) {

printf("File is not open for writing.\n");

} else {

printf("File is open for writing.\n");

// Check if all contents have been written until the last block

if (fflush(file) != 0) {

printf("Error flushing file.\n");

} else if (fseek(file, 0L, SEEK\_END) != 0) {

printf("Error seeking to end of file.\n");

} else {

long last\_block\_size = ftell(file) % 512;

if (last\_block\_size != 0) {

printf("File has not been written until the last block.\n");

} else {

printf("File has been written until the last block.\n");

}

}

}

fclose(file);

// Move the file to the specified directory

char new\_filename[strlen(directory) + strlen(filename) + 2];

sprintf(new\_filename, "%s/%s", directory, filename);

if (rename(filename, new\_filename) != 0) {

perror("Error moving file");

return 1;

}

return 0;

}

//////////////////

Scp response

If you want to use the libssh library to execute an scp command and capture its response, you can use the ssh\_channel\_open\_session() and ssh\_channel\_request\_exec() functions to open an SSH channel and execute the scp command, respectively. Here's an example program that demonstrates how to do this:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <libssh/libssh.h>

int main() {

char \*filename = "file\_to\_send";

char \*destination = "user@server:/path/to/destination/";

// Initialize libssh

ssh\_session session = ssh\_new();

if (session == NULL) {

perror("Error initializing libssh");

return 1;

}

// Connect to the SSH server

ssh\_options\_set(session, SSH\_OPTIONS\_HOST, "server");

ssh\_options\_set(session, SSH\_OPTIONS\_USER, "user");

int rc = ssh\_connect(session);

if (rc != SSH\_OK) {

fprintf(stderr, "Error connecting to SSH server: %s\n", ssh\_get\_error(session));

ssh\_free(session);

return 1;

}

// Authenticate with the server using a password

rc = ssh\_userauth\_password(session, NULL, "password");

if (rc != SSH\_AUTH\_SUCCESS) {

fprintf(stderr, "Error authenticating with SSH server: %s\n", ssh\_get\_error(session));

ssh\_disconnect(session);

ssh\_free(session);

return 1;

}

// Open an SSH channel

ssh\_channel channel = ssh\_channel\_new(session);

if (channel == NULL) {

fprintf(stderr, "Error creating SSH channel: %s\n", ssh\_get\_error(session));

ssh\_disconnect(session);

ssh\_free(session);

return 1;

}

// Request a pty for the channel

rc = ssh\_channel\_request\_pty(channel);

if (rc != SSH\_OK) {

fprintf(stderr, "Error requesting pty for SSH channel: %s\n", ssh\_get\_error(session));

ssh\_channel\_free(channel);

ssh\_disconnect(session);

ssh\_free(session);

return 1;

}

// Open the channel for execution

rc = ssh\_channel\_open\_session(channel);

if (rc != SSH\_OK) {

fprintf(stderr, "Error opening SSH channel: %s\n", ssh\_get\_error(session));

ssh\_channel\_free(channel);

ssh\_disconnect(session);

ssh\_free(session);

return 1;

}

// Build the scp command

char command[256];

snprintf(command, sizeof(command), "scp %s %s", filename, destination);

// Execute the command on the channel

rc = ssh\_channel\_request\_exec(channel, command);

if (rc != SSH\_OK) {

fprintf(stderr, "Error executing command on SSH channel: %s\n", ssh\_get\_error(session));

ssh\_channel\_free(channel);

ssh\_disconnect(session);

ssh\_free(session);

return 1;

}

// Read the output of the command from the channel and print it

char buffer[256];

ssize\_t nread;

while ((nread = ssh\_channel\_read(channel, buffer, sizeof(buffer), 0)) > 0) {

fwrite(buffer, sizeof(char), nread, stdout);

}

// Close the channel and disconnect from the server

ssh\_channel\_send\_eof(channel);

ssh\_channel\_close(channel);

ssh\_channel\_free(channel);

ssh\_disconnect(session);

ssh\_free(session);

return 0;

}