#include <stdio.h>

#include <libssh/libssh.h>

#include <string.h>

int main() {

ssh\_session session;

ssh\_scp scp;

int port = 22;

int rc;

char \*remote\_path = "/remote/path/file.txt";

char \*local\_path = "/local/path/file.txt";

// Initialize libssh

ssh\_init();

// Connect to the remote host

session = ssh\_new();

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

ssh\_options\_set(session, SSH\_OPTIONS\_PORT, &port);

rc = ssh\_connect(session);

if (rc != SSH\_OK) {

printf("Error: Could not connect to remote host (%s)\n", ssh\_get\_error(session));

ssh\_free(session);

return 1;

}

// Authenticate using password authentication

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

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

printf("Error: Could not authenticate to remote host (%s)\n", ssh\_get\_error(session));

ssh\_disconnect(session);

ssh\_free(session);

return 1;

}

// Create an SCP object

scp = ssh\_scp\_new(session, SSH\_SCP\_WRITE, remote\_path);

if (scp == NULL) {

printf("Error: Could not create SCP object (%s)\n", ssh\_get\_error(session));

ssh\_disconnect(session);

ssh\_free(session);

return 1;

}

// Initialize the SCP transfer

rc = ssh\_scp\_init(scp);

if (rc != SSH\_OK) {

printf("Error: Could not initialize SCP transfer (%s)\n", ssh\_get\_error(session));

ssh\_scp\_free(scp);

ssh\_disconnect(session);

ssh\_free(session);

return 1;

}

// Open the local file for reading

FILE \*local\_file = fopen(local\_path, "r");

if (local\_file == NULL) {

printf("Error: Could not open local file for reading (%s)\n", strerror(errno));

ssh\_scp\_close(scp);

ssh\_scp\_free(scp);

ssh\_disconnect(session);

ssh\_free(session);

return 1;

}

// Transfer the file

char buffer[4096];

size\_t nread;

int total\_bytes\_written = 0;

while ((nread = fread(buffer, 1, sizeof(buffer), local\_file)) > 0) {

rc = ssh\_scp\_write(scp, buffer, nread);

if (rc != SSH\_OK) {

printf("Error: Could not write to remote file (%s)\n", ssh\_get\_error(session));

fclose(local\_file);

ssh\_scp\_close(scp);

ssh\_scp\_free(scp);

ssh\_disconnect(session);

ssh\_free(session);

return 1;

}

total\_bytes\_written += nread;

}

// Close the local and remote files

fclose(local\_file);

ssh\_scp\_close(scp);

// Cleanup

ssh\_scp\_free(scp);

ssh\_disconnect(session);

ssh\_free(session);

printf("Transferred %d bytes successfully\n", total\_bytes\_written);

return 0;

}

This program uses the ssh\_scp\_new() and ssh\_scp\_init() functions to create and initialize an SCP object, respectively. It then opens the local file for reading and transfers it using the ssh\_scp\_write() function.

To transfer a file using libssh scp in C with both local and remote file handlers, the following steps can be followed:

Open the local file using fopen() and get the local file handler.

Open an SSH session using ssh\_new() and ssh\_connect().

Authenticate the SSH session using ssh\_userauth\_password() or ssh\_userauth\_publickey\_auto().

Open an SCP session using ssh\_scp\_new() and ssh\_scp\_init().

Send the file to the remote server using ssh\_scp\_push\_file() and the local file handler.

Open the remote file using ssh\_scp\_request\_get\_filename() and get the remote file handler.

Read the data from the local file using fread() and write it to the remote file using ssh\_scp\_write() and the remote file handler.

Close the local file using fclose().

Close the remote file using ssh\_scp\_close\_file() and the remote file handler.

Close the SCP session using ssh\_scp\_close().

Close the SSH session using ssh\_disconnect() and ssh\_free().

Here's an example code snippet that demonstrates how to transfer a file using libssh scp in C with both local and remote file handlers:

#include <libssh/libssh.h>

#include <stdio.h>

int main() {

ssh\_session session;

ssh\_scp scp;

FILE \*local\_file;

ssh\_scp\_file\_info file\_info;

FILE \*remote\_file;

char buffer;

size\_t bytes\_read;

int rc;

// Open the local file

local\_file = fopen("/path/to/local/file", "rb");

if (local\_file == NULL) {

printf("Error opening local file\n");

return 1;

}

// Open an SSH session

session = ssh\_new();

ssh\_options\_set(session, SSH\_OPTIONS\_HOST, "example.com");

ssh\_connect(session);

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

// Open an SCP session

scp = ssh\_scp\_new(session, SSH\_SCP\_WRITE, "/path/to/remote/file");

ssh\_scp\_init(scp);

// Send the file to the remote server

rc = ssh\_scp\_push\_file(scp, "filename", filesize, 0644);

if (rc != SSH\_OK) {

printf("Error sending file: %s\n", ssh\_get\_error(session));

fclose(local\_file);

return 1;

}

// Open the remote file

file\_info = ssh\_scp\_request\_get(scp);

remote\_file = fdopen(file\_info->fd, "wb");

// Read the data from the local file and write it to the remote file

while ((bytes\_read = fread(buffer, 1, sizeof(buffer), local\_file)) > 0) {

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

}

// Close the local file

fclose(local\_file);

// Close the remote file

ssh\_scp\_close\_file(scp, file\_info);

// Close the SCP session

ssh\_scp\_close(scp);

ssh\_scp\_free(scp);

// Close the SSH session

ssh\_disconnect(session);

ssh\_free(session);

return 0;

}

In this example, the local file is opened using fopen() and the local file handler is obtained. An SSH session is opened using ssh\_new() and ssh\_connect(). The session is authenticated using ssh\_userauth\_password(). An SCP session is opened using ssh\_scp\_new() and ssh\_scp\_init(). The file is sent to the remote server using ssh\_scp\_push\_file() and the local file handler. The remote file is opened using ssh\_scp\_request\_get\_filename() and the remote file handler is obtained. The data is read from the local file using fread() and written to the remote file using fwrite() and the remote file handler. The local file is closed using fclose(). The remote file is closed using ssh\_scp\_close\_file() and the remote file handler. The SCP session is closed using ssh\_scp\_close() and ssh\_scp\_free(). The SSH session is closed using ssh\_disconnect() and ssh\_free().

**https://api.libssh.org/stable/libssh\_tutor\_scp.html**

`` #include <stdio.h>

#include <stdlib.h>

#include <libssh/libssh.h>

int main() {

ssh\_session session;

ssh\_scp scp\_write;

int rc;

// Initialize a new SSH session

session = ssh\_new();

if (session == NULL) {

fprintf(stderr, "Error: Could not create SSH session\n");

return 1;

}

// Set the options for the SSH session

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

ssh\_options\_set(session, SSH\_OPTIONS\_PORT, 22);

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

// Connect to the SSH server

rc = ssh\_connect(session);

if (rc != SSH\_OK) {

fprintf(stderr, "Error: Could not connect to SSH server (%s)\n", ssh\_get\_error(session));

ssh\_free(session);

return 1;

}

// Authenticate with the SSH server using a password

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

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

fprintf(stderr, "Error: Could not authenticate with SSH server (%s)\n", ssh\_get\_error(session));

ssh\_disconnect(session);

ssh\_free(session);

return 1;

}

// Send multiple files using SCP

char \*local\_files[] = {"file1.txt", "file2.txt", "file3.txt"};

char \*remote\_path = "/remote/directory/";

int num\_files = sizeof(local\_files) / sizeof(local\_files[0]);

int i, file\_handler;

for (i = 0; i < num\_files; i++) {

// Open the local file for reading

FILE \*local\_file;

local\_file = fopen(local\_files[i], "r");

if (local\_file == NULL) {

fprintf(stderr, "Error: Could not open local file for reading\n");

ssh\_disconnect(session);

ssh\_free(session);

return 1;

}

// Create a new SCP channel for writing

scp\_write = ssh\_scp\_new(session, SSH\_SCP\_WRITE, remote\_path);

if (scp\_write == NULL) {

fprintf(stderr, "Error: Could not create SCP channel (%s)\n", ssh\_get\_error(session));

fclose(local\_file);

ssh\_disconnect(session);

ssh\_free(session);

return 1;

}

// Initialize the SCP channel for writing

rc = ssh\_scp\_init(scp\_write);

if (rc != SSH\_OK) {

fprintf(stderr, "Error: Could not initialize SCP channel (%s)\n", ssh\_get\_error(session));

fclose(local\_file);

ssh\_scp\_close(scp\_write);

ssh\_scp\_free(scp\_write);

ssh\_disconnect(session);

ssh\_free(session);

return 1;

}

// Write to the remote file

int nbytes;

char write\_buffer[1024];

while ((nbytes = fread(write\_buffer, 1, sizeof(write\_buffer), local\_file)) > 0) {

if (ssh\_scp\_write(scp\_write, write\_buffer, nbytes) != SSH\_OK) {

fprintf(stderr, "Error: Could not write to remote file (%s)\n", ssh\_get\_error(session));

fclose(local\_file);

ssh\_scp\_close(scp\_write);

ssh\_scp\_free(scp\_write);

ssh\_disconnect(session);

ssh\_free(session);

return 1;

}

}

// Close the local and remote files

fclose(local\_file);

// Check for errors while writing to remote file

file\_handler = ssh\_scp\_leave\_directory(scp\_write);

if (file\_handler != SSH\_OK) {

fprintf(stderr, "Error: Could not leave SCP directory (%s)\n", ssh\_get\_error(session));

ssh\_scp\_close(scp\_write);

ssh\_scp\_free(scp\_write);

ssh\_disconnect(session);

ssh\_free(session);

return 1;

}

// Free the SCP channel

ssh\_scp\_close(scp\_write);

ssh\_scp\_free(scp\_write);

// Print an acknowledgement message

printf("File %s sent successfully\n", local\_files[i]);

}

// Disconnect from the SSH server and free the SSH session

ssh\_disconnect(session);

ssh\_free(session);

// Return 0 to indicate success

return 0;

}

In this example, we are sending multiple files from the client to the server using SCP. We loop through each file and perform the following steps:

1. Open the local file for reading

2. Create a new SCP channel for writing

3. Initialize the SCP channel for writing

4. Write to the remote file using the `ssh\_scp\_write()` function

5. Close the local and remote files

6. Check for errors while writing to the remote file using `ssh\_scp\_leave\_directory()`

7. Free the SCP channel

8. Print an acknowledgement message

If there is an error during any step, the program will print an error message and return a non-zero value to indicate failure. If all steps are successful, the program will print a success message and return a zero value to indicate success.

--

In the above code, the program first establishes an SSH session and authenticates with the server using a password. Then, it loops through an array of local files and sends each one to the remote server using SCP.

For each file, the program first opens the local file for reading and creates a new SCP channel for writing. It then initializes the SCP channel and writes the contents of the local file to the remote file using `ssh\_scp\_write()`. Finally, it closes the local and remote files and checks for errors while closing the SCP channel using `ssh\_scp\_close()`.

If any error occurs during the file transfer, the program prints an error message and returns a non-zero exit code. If all files are transferred successfully, the program prints a success message and returns a zero exit code.

[**https://github.com/xbmc/libssh/blob/master/src/scp.c**](https://github.com/xbmc/libssh/blob/master/src/scp.c)

@@

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <libssh/libssh.h>

#include <dirent.h>

#include <sys/stat.h>

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

{

// Check for correct number of command line arguments

if (argc < 3) {

printf("Usage: %s <directory\_path> <remote\_dir>\n", argv[0]);

exit(EXIT\_FAILURE);

}

// Remote server details

char \*remote\_host = "192.168.0.100";

int remote\_port = 22;

char \*remote\_username = "john";

char \*remote\_password = "secret";

char \*remote\_dir = argv[2];

// Local directory path and file list

char \*local\_dir = argv[1];

char \*\*local\_files;

int num\_files = 0;

// Open local directory and count files

DIR \*dir = opendir(local\_dir);

if (dir == NULL) {

printf("Error: Failed to open local directory '%s'.\n", local\_dir);

exit(EXIT\_FAILURE);

}

struct dirent \*dp;

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

// Ignore directories and hidden files

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

continue;

}

num\_files++;

}

closedir(dir);

// Allocate memory for local file paths array

local\_files = (char \*\*) malloc(sizeof(char \*) \* num\_files);

// Open local directory and store file paths in array

dir = opendir(local\_dir);

if (dir == NULL) {

printf("Error: Failed to open local directory '%s'.\n", local\_dir);

exit(EXIT\_FAILURE);

}

int i = 0;

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

// Ignore directories and hidden files

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

continue;

}

char \*file\_path = (char \*) malloc(sizeof(char) \* (strlen(local\_dir) + strlen(dp->d\_name) + 2));

sprintf(file\_path, "%s/%s", local\_dir, dp->d\_name);

local\_files[i++] = file\_path;

}

closedir(dir);

// Initialize SSH session and channel

ssh\_session session;

ssh\_channel channel;

int rc;

session = ssh\_new();

if (session == NULL) {

printf("Error: Failed to create SSH session.\n");

exit(EXIT\_FAILURE);

}

// Set SSH options

ssh\_options\_set(session, SSH\_OPTIONS\_HOST, remote\_host);

ssh\_options\_set(session, SSH\_OPTIONS\_PORT, &remote\_port);

ssh\_options\_set(session, SSH\_OPTIONS\_USER, remote\_username);

// Connect to remote server

rc = ssh\_connect(session);

if (rc != SSH\_OK) {

printf("Error: Failed to connect to remote server.\n%s\n", ssh\_get\_error(session));

ssh\_free(session);

exit(EXIT\_FAILURE);

}

// Authenticate with password

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

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

printf("Error: Failed to authenticate with remote server.\n%s\n", ssh\_get\_error(session));

ssh\_disconnect(session);

ssh\_free(session);

exit(EXIT\_FAILURE);

}

// Open SCP channel for file transfer

channel = ssh\_channel\_new(session);

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

if (rc != SSH\_OK) {

printf("Error: Failed to open SSH session for SCP file transfer.\n%s\n", ssh\_get\_error(session));

ssh\_disconnect(session);

ssh\_free(session);

exit(EXIT\_FAILURE);

}

// Iterate over local files and transfer to remote server

for (i = 0; i < num\_files; i++) {

// Open local file for reading

FILE \*local\_file = fopen(local\_files[i], "rb");

if (local\_file == NULL) {

printf("Error: Failed to open local file '%s' for reading.\n", local\_files[i]);

continue;

}

// Get size of local file

struct stat local\_file\_info;

if (stat(local\_files[i], &local\_file\_info) < 0) {

printf("Error: Failed to get size of local file '%s'.\n", local\_files[i]);

fclose(local\_file);

continue;

}

// Send SCP command to create remote file

char \*scp\_command = (char \*) malloc(sizeof(char) \* (strlen(local\_files[i]) + strlen(remote\_dir) + 32));

sprintf(scp\_command, "scp -t %s/%s", remote\_dir, strrchr(local\_files[i], '/') ? strrchr(local\_files[i], '/') + 1 : local\_files[i]);

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

if (rc != SSH\_OK) {

printf("Error: Failed to execute SCP command for remote file '%s/%s'.\n%s\n", remote\_dir, strrchr(local\_files[i], '/') ? strrchr(local\_files[i], '/') + 1 : local\_files[i], ssh\_get\_error(session));

fclose(local\_file);

continue;

}

// Read SCP response for remote file creation

char buffer[256];

rc = ssh\_channel\_read(channel, buffer, sizeof(buffer), SSH\_CHANNEL\_EXTENDED\_DATA);

if (rc < 0) {

printf("Error: Failed to read SCP response for remote file '%s/%s'.\n%s\n", remote\_dir, strrchr(local\_files[i], '/') ? strrchr(local\_files[i], '/') + 1 : local\_files[i], ssh\_get\_error(session));

fclose(local\_file);

continue;

}

// Send local file contents to remote file

size\_t bytes\_sent = 0;

while (bytes\_sent < local\_file\_info.st\_size) {

size\_t bytes\_remaining = local\_file\_info.st\_size - bytes\_sent;

size\_t block\_size = bytes\_remaining < sizeof(buffer) ? bytes\_remaining : sizeof(buffer);

size\_t bytes\_read = fread(buffer, 1, block\_size, local\_file);

if (bytes\_read <= 0) {

printf("Error: Failed to read from local file '%s'.\n", local\_files[i]);

break;

}

rc = ssh\_channel\_write(channel, buffer, bytes\_read);

if (rc < 0) {

printf("Error: Failed to send data for remote file '%s/%s'.\n%s\n", remote\_dir, strrchr(local\_files[i], '/') ? strrchr(local\_files[i], '/') + 1 : local\_files[i], ssh\_get\_error(session));

break;

}

bytes\_sent += bytes\_read;

}

fclose(local\_file);

// Send EOF to signal end of file transfer

rc = ssh\_channel\_send\_eof(channel);

if (rc != SSH\_OK) {

printf("Error: Failed to send EOF for remote file '%s/%s'.\n%s\n", remote\_dir, strrchr(local\_files[i], '/') ? strrchr(local\_files[i], '/') + 1 : local\_files[i], ssh\_get\_error(session));

continue;

}

// Wait for remote SCP command to exit

rc = ssh\_channel\_wait\_for\_exit(channel);

if (rc != SSH\_OK) {

printf("Error: Failed to wait for remote SCP command to exit for file '%s/%s'.\n%s\n", remote\_dir, strrchr(local\_files[i], '/') ? strrchr(local\_files[i], '/') + 1 : local\_files[i], ssh\_get\_error(session));

continue;

}

// Check SCP exit status

int exit\_status = ssh\_channel\_get\_exit\_status(channel);

if (exit\_status != 0) {

printf("Error: Remote SCP command for file '%s/%s' exited with status %d.\n", remote\_dir, strrchr(local\_files[i], '/') ? strrchr(local\_files[i], '/') + 1 : local\_files[i], exit\_status);

continue;

}

printf("Successfully transferred local file '%s' to remote directory '%s'.\n", local\_files[i], remote\_dir);

}

// Cleanup

ssh\_channel\_free(channel);

ssh\_disconnect(session);

ssh\_free(session);

return 0;

}

Entry point for SCP file transfer program.

Accepts two command-line arguments: remote server address and remote directory path.

Additional arguments should be local file paths to transfer to the remote server.

\*/

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

if (argc < 4) {

printf("Usage: %s <remote-server> <remote-dir> <local-file> [local-file] ...\n", argv[0]);

return 1;

}

char \*remote\_server = argv[1];

char \*remote\_dir = argv[2];

char \*\*local\_files = argv + 3;

int num\_files = argc - 3;

return scp\_transfer\_files(remote\_server, remote\_dir, local\_files, num\_files);

}

This program should allow you to transfer one or more files from a local machine to a remote server using the SCP protocol. You can run it by compiling the source code and then executing the resulting binary, providing the necessary command-line arguments for the remote server address, remote directory path, and local file paths to transfer.

int send\_files\_via\_scp(ssh\_session session, const char \*local\_dir, const char \*remote\_dir) {

// Open SSH channel for SCP file transfer

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

if (channel == NULL) {

printf("Error: Failed to create SSH channel for SCP file transfer.\n");

return -1;

}

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

if (rc != SSH\_OK) {

printf("Error: Failed to open SSH session for SCP file transfer.\n%s\n", ssh\_get\_error(session));

ssh\_channel\_free(channel);

return -1;

}

// Iterate over local files and transfer to remote server

int num\_files\_sent = 0;

DIR \*dir;

struct dirent \*ent;

if ((dir = opendir(local\_dir)) != NULL) {

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

// Skip current and parent directory entries

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

continue;

}

// Construct local and remote file paths

char local\_file\_path[PATH\_MAX];

char remote\_file\_path[PATH\_MAX];

snprintf(local\_file\_path, PATH\_MAX, "%s/%s", local\_dir, ent->d\_name);

snprintf(remote\_file\_path, PATH\_MAX, "%s/%s", remote\_dir, ent->d\_name);

// Open local file for reading

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

if (local\_file == NULL) {

printf("Error: Failed to open local file '%s' for reading.\n", local\_file\_path);

continue;

}

// Get size of local file

struct stat local\_file\_info;

if (fstat(fileno(local\_file), &local\_file\_info) < 0) {

printf("Error: Failed to get size of local file '%s'.\n", local\_file\_path);

fclose(local\_file);

continue;

}

// Send SCP command to create remote file

char \*scp\_command = (char \*) malloc(sizeof(char) \* (strlen(remote\_file\_path) + 32));

sprintf(scp\_command, "scp -t %s", remote\_file\_path);

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

if (rc != SSH\_OK) {

printf("Error: Failed to execute SCP command for remote file '%s'.\n%s\n", remote\_file\_path, ssh\_get\_error(session));

fclose(local\_file);

continue;

}

// Read SCP response for remote file creation

char buffer[256];

rc = ssh\_channel\_read(channel, buffer, sizeof(buffer), SSH\_CHANNEL\_EXTENDED\_DATA);

if (rc < 0) {

printf("Error: Failed to read SCP response for remote file '%s'.\n%s\n", remote\_file\_path, ssh\_get\_error(session));

fclose(local\_file);

continue;

}

// Check if remote server acknowledged file transfer

if (buffer[0] != SSH\_SCP\_RESPONSE\_OK) {

printf("Error: Failed to receive acknowledgement from remote server for file '%s'.\n", remote\_file\_path);

fclose(local\_file);

continue;

}

// Send local file contents to remote file

size\_t bytes\_sent = 0;

while (bytes\_sent < local\_file\_info.st\_size) {

size\_t bytes\_remaining = local\_file\_info.st\_size - bytes\_sent; // Determine size of next chunk to send

size\_t chunk\_size = (bytes\_remaining > SCP\_MAX\_BYTES) ? SCP\_MAX\_BYTES : bytes\_remaining;

// Read chunk from local file

char \*chunk = (char \*) malloc(sizeof(char) \* chunk\_size);

size\_t bytes\_read = fread(chunk, sizeof(char), chunk\_size, local\_file);

if (bytes\_read < chunk\_size && !feof(local\_file)) {

printf("Error: Failed to read chunk from local file '%s'.\n", local\_file\_path);

free(chunk);

fclose(local\_file);

continue;

}

// Send chunk to remote server

rc = ssh\_channel\_write(channel, chunk, bytes\_read);

if (rc < 0) {

printf("Error: Failed to send chunk to remote file '%s'.\n%s\n", remote\_file\_path, ssh\_get\_error(session));

free(chunk);

fclose(local\_file);

continue;

}

bytes\_sent += bytes\_read;

free(chunk);

}

// Send null byte to signal end of file transfer

char null\_byte = '\0';

rc = ssh\_channel\_write(channel, &null\_byte, sizeof(null\_byte));

if (rc < 0) {

printf("Error: Failed to signal end of file transfer for remote file '%s'.\n%s\n", remote\_file\_path, ssh\_get\_error(session));

fclose(local\_file);

continue;

}

// Read SCP response for end of file transfer

rc = ssh\_channel\_read(channel, buffer, sizeof(buffer), SSH\_CHANNEL\_EXTENDED\_DATA);

if (rc < 0) {

printf("Error: Failed to read SCP response for remote file '%s'.\n%s\n", remote\_file\_path, ssh\_get\_error(session));

fclose(local\_file);

continue;

}

// Check if remote server acknowledged end of file transfer

if (buffer[0] != SSH\_SCP\_RESPONSE\_OK) {

printf("Error: Failed to receive acknowledgement from remote server for end of file transfer for '%s'.\n", remote\_file\_path);

fclose(local\_file);

continue;

}

fclose(local\_file);

num\_files\_sent++;

}

closedir(dir);

} else {

printf("Error: Failed to open local directory '%s'.\n", local\_dir);

ssh\_channel\_free(channel);

return -1;

}

// Close SCP channel and free resources

ssh\_channel\_send\_eof(channel);

ssh\_channel\_close(channel);

ssh\_channel\_free(channel);

return num\_files\_sent;

}