#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <dirent.h>

#include <sys/stat.h>

#include <libssh/libssh.h>

#include <libssh/sftp.h>

#define SSH\_SCP\_RESPONSE\_OK 0

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

char file\_buffer[local\_file\_info.st\_size];

size\_t bytes\_read = fread(file\_buffer, 1, local\_file\_info.st\_size, local\_file);

if (bytes\_read != local\_file\_info.st\_size) {

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

fclose(local\_file);

continue;

}

size\_t bytes\_sent = 0;

while (bytes\_sent < bytes\_read) {

// Determine number of bytes to send in this iteration

size\_t bytes\_to\_send = bytes\_read - bytes\_sent;

if (bytes\_to\_send > sizeof(buffer)) {

bytes\_to\_send = sizeof(buffer);

}

// Send bytes to remote file

rc = ssh\_channel\_write(channel, &file\_buffer[bytes\_sent], bytes\_to\_send);

if (rc < 0) {

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

fclose(local\_file);

continue;

}

bytes\_sent += rc;

}

fclose(local\_file);

num\_files\_sent++;

}

closedir(dir);

ssh\_channel\_send\_eof(channel);

ssh\_channel\_close(channel);

ssh\_channel\_free(channel);

return num\_files\_sent;

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <dirent.h>

#include <sys/stat.h>

#include <libssh/libssh.h>

#include <libssh/sftp.h>

#define SSH\_SCP\_RESPONSE\_OK 0

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\_read;

char file\_contents[4096];

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

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

if (rc != SSH\_OK) {

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

break;

}

}

fclose(local\_file);

// Send null byte to indicate end of file transfer

char null\_byte = '\0';

rc = ssh\_channel\_write(channel, &null\_byte, 1);

if (rc != SSH\_OK) {

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

continue;

}

num\_files\_sent++;

}

closedir(dir);

} else {

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

ssh\_channel\_free(channel);

return -1;

}

ssh\_channel\_send\_eof(channel);

ssh\_channel\_close(channel);

ssh\_channel\_free(channel);

printf("Sent %d files to remote directory '%s'.\n", num\_files\_sent, remote\_dir);

return num\_files\_sent;

}

This function will iterate over each file in the local directory specified by local\_dir, and send it to the remote directory specified by remote\_dir. It will open the local file for reading, get the size of the file, and then send an SCP command to the remote server to create the file. It will then read the response from the server to verify that the file was created successfully, and then send the contents of the local file to the remote file. Once the file transfer is complete, it will close the local file and move on to the next file in the directory. Finally, it will print out the number of files that were successfully sent to the remote directory.