When sending a file using SCP using libssh, the response will depend on whether the file transfer was successful or not.

If the file transfer was successful, you will typically get a response similar to the following:

/path/to/local/file 100% 1024 50.0KB/s 00:00

This response indicates that the file was transferred successfully, and it shows the path to the local file, the percentage of the file that was transferred, the file size, the transfer rate, and the transfer time.

If the file transfer failed, you may get one of the following error messages:

"Permission denied": This error message indicates that the user does not have permission to access the file or directory on the remote server.

"No such file or directory": This error message indicates that the file or directory on the remote server does not exist.

"Connection reset by peer": This error message indicates that the connection to the remote server was reset before the file transfer could be completed.

In addition to these error messages, there may be other error messages that could indicate specific issues with the file transfer, such as network errors or file system errors. It's a good practice to check the output of the SCP command to make sure that the file was transferred successfully and to handle any errors that may occur during the transfer.

```````````````````````````

When using libssh to send a file using SCP, the SCP command will return a message indicating success or failure.

Here are the possible responses you may receive when sending a file using SCP with libssh:

If the file transfer is successful, you will receive no output from the SCP command.

If the destination file already exists, you will receive an error message that says "file exists" or "not overwritten".

If there is a permission error, you will receive an error message that says "permission denied".

If the destination directory does not exist, you will receive an error message that says "No such file or directory".

If there is a network error, you may receive an error message that says "lost connection" or "broken pipe".

You can capture and parse these responses to determine whether the file transfer was successful or not. If there is an error, you can use the error message to diagnose the issue and take appropriate action.

When using libssh to transfer a file using SCP, you can use the ssh\_channel\_is\_closed() function to check the status of the file transfer.

Here is an example of how you could use this function:

#include <libssh/libssh.h>

// Initialize libssh and connect to the server

// Open an SCP channel for file transfer

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

ssh\_channel\_open\_session(scp\_channel);

// Start the SCP transfer

ssh\_channel\_request\_exec(scp\_channel, "scp -t /path/to/remote/file");

// Write the contents of the local file to the SCP channel

// ...

// Check if the SCP channel is closed

while (!ssh\_channel\_is\_closed(scp\_channel)) {

// Wait for the channel to close

}

// Check the exit status of the SCP command

int scp\_exit\_status = ssh\_channel\_get\_exit\_status(scp\_channel);

if (scp\_exit\_status == 0) {

// File transfer was successful

} else {

// There was an error during the file transfer

}

// Close the SCP channel and disconnect from the server

ssh\_channel\_send\_eof(scp\_channel);

ssh\_channel\_close(scp\_channel);

ssh\_channel\_free(scp\_channel);

ssh\_disconnect(session);

ssh\_free(session);

In this example, the ssh\_channel\_is\_closed() function is used to wait for the SCP channel to close before checking the exit status of the SCP command. Once the channel is closed, the ssh\_channel\_get\_exit\_status() function is used to retrieve the exit status of the SCP command. If the exit status is 0, the file transfer was successful. If the exit status is non-zero, there was an error during the file transfer.

When using libssh to transfer files with SCP, you can use the scp\_send() or scp\_recv() function to initiate the file transfer. These functions return a file handle, which can be used to check the status of the file transfer.

Here is an example of how to use scp\_send() to transfer a file and check the file handle status:

#include <libssh/libssh.h>

#include <stdlib.h>

int main() {

ssh\_session session;

ssh\_scp scp;

ssh\_channel channel;

int rc;

session = ssh\_new();

// set up session options and connect to server

// ...

// initiate the SCP transfer and get the file handle

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

if (scp == NULL) {

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

exit(EXIT\_FAILURE);

}

rc = ssh\_scp\_init(scp);

if (rc != SSH\_OK) {

fprintf(stderr, "Error initializing SCP session: %s\n", ssh\_get\_error(session));

exit(EXIT\_FAILURE);

}

const char \*local\_path = "/path/to/local/file";

struct stat st;

stat(local\_path, &st);

ssh\_scp\_push\_file(scp, local\_path, st.st\_size, S\_IRUSR | S\_IWUSR);

// get the file handle and check its status

channel = ssh\_scp\_get\_channel(scp);

if (channel == NULL) {

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

exit(EXIT\_FAILURE);

}

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

if (status != SSH\_OK) {

fprintf(stderr, "Error transferring file: %s\n", ssh\_get\_error(session));

exit(EXIT\_FAILURE);

}

ssh\_scp\_close(scp);

ssh\_scp\_free(scp);

ssh\_disconnect(session);

ssh\_free(session);

return 0;

}

In this example, ssh\_scp\_push\_file() is used to transfer a file to the remote server. Once the transfer is complete, the ssh\_scp\_get\_channel() function is used to get the file handle, which is stored in the channel variable. Finally, ssh\_channel\_get\_exit\_status() is used to check the status of the file transfer. If the status is not SSH\_OK, an error has occurred and the program will exit with an error message.

To get the status of the file transfer using scp\_send() or scp\_recv() functions in libssh, you can use the ssh\_scp\_pull\_request() and ssh\_scp\_push\_request() functions respectively.

Here is an example of how to use ssh\_scp\_pull\_request() with scp\_recv() to check the status of the file transfer:

#include <libssh/libssh.h>

#include <stdlib.h>

int main() {

ssh\_session session;

ssh\_scp scp;

ssh\_channel channel;

int rc;

session = ssh\_new();

// set up session options and connect to server

// ...

// initiate the SCP transfer and get the file handle

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

if (scp == NULL) {

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

exit(EXIT\_FAILURE);

}

rc = ssh\_scp\_init(scp);

if (rc != SSH\_OK) {

fprintf(stderr, "Error initializing SCP session: %s\n", ssh\_get\_error(session));

exit(EXIT\_FAILURE);

}

// receive the file and get the file handle

rc = ssh\_scp\_pull\_request(scp);

if (rc != SSH\_SCP\_REQUEST\_NEWFILE) {

fprintf(stderr, "Error requesting file: %s\n", ssh\_get\_error(session));

exit(EXIT\_FAILURE);

}

channel = ssh\_scp\_accept\_request(scp);

if (channel == NULL) {

fprintf(stderr, "Error accepting SCP request: %s\n", ssh\_get\_error(session));

exit(EXIT\_FAILURE);

}

// read from the file handle and check its status

char buffer[1024];

size\_t nbytes;

do {

nbytes = ssh\_channel\_read(channel, buffer, sizeof(buffer), 0);

if (nbytes < 0) {

fprintf(stderr, "Error reading from SCP channel: %s\n", ssh\_get\_error(session));

exit(EXIT\_FAILURE);

}

} while (nbytes > 0);

int status = ssh\_scp\_pull\_request(scp);

if (status != SSH\_SCP\_REQUEST\_EOF) {

fprintf(stderr, "Error transferring file: %s\n", ssh\_get\_error(session));

exit(EXIT\_FAILURE);

}

ssh\_scp\_free(scp);

ssh\_disconnect(session);

ssh\_free(session);

return 0;

}

In this example, ssh\_scp\_pull\_request() is used to receive the file from the remote server. Once the transfer is complete, ssh\_scp\_pull\_request() is called again to check the status of the file transfer. If the status is SSH\_SCP\_REQUEST\_EOF, the transfer was successful and the program will exit with a success status. If the status is not SSH\_SCP\_REQUEST\_EOF, an error has occurred and the program will exit with an error message.

Here's an example C program that demonstrates how to get the file handler status for file transfer on the client side using the libssh library:

c

Copy code

#include <stdio.h>

#include <libssh/libssh.h>

int main() {

ssh\_session session;

ssh\_scp scp;

int status;

// create a new SSH session

session = ssh\_new();

if (session == NULL) {

fprintf(stderr, "Error creating SSH session\n");

return 1;

}

// set the SSH options and connect to the remote host

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

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

if (ssh\_connect(session) != SSH\_OK) {

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

return 1;

}

// authenticate with the remote host using a password

if (ssh\_userauth\_password(session, NULL, "password") != SSH\_AUTH\_SUCCESS) {

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

return 1;

}

// create an SCP session and transfer a file

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

if (scp == NULL) {

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

return 1;

}

if (ssh\_scp\_init(scp) != SSH\_OK) {

fprintf(stderr, "Error initializing SCP session: %s\n", ssh\_get\_error(session));

return 1;

}

if (ssh\_scp\_push\_file(scp, "local-file", 0) != SSH\_OK) {

fprintf(stderr, "Error transferring file: %s\n", ssh\_get\_error(session));

return 1;

}

// check the file handler status

status = ssh\_scp\_write(scp, "data", 4);

if (status != SSH\_OK) {

fprintf(stderr, "Error writing to file: %s\n", ssh\_get\_error(session));

return 1;

}

// close the SCP session and SSH session

ssh\_scp\_close(scp);

ssh\_scp\_free(scp);

ssh\_disconnect(session);

ssh\_free(session);

return 0;

}

In this example, we create a new SSH session using ssh\_new() and set the SSH options to connect to the remote host using ssh\_options\_set(). We then connect to the remote host using ssh\_connect() and authenticate with the remote host using a password using ssh\_userauth\_password().

Next, we create an SCP session using ssh\_scp\_new() and transfer a file using ssh\_scp\_push\_file(). After transferring the file, we check the file handler status using ssh\_scp\_write() to write some data to the file. If an error occurs, ssh\_scp\_write() returns a non-zero value, and we print an error message and return a non-zero value from main(). Finally, we close the SCP session using ssh\_scp\_close() and ssh\_scp\_free() and the SSH session using ssh\_disconnect() and ssh\_free().