Chapter 6: The SCP subsystem

The SCP subsystem

The SCP subsystem has far less functionality than the SFTP subsystem. However, if you only need to copy files from and to the remote system, it does its job.

Opening and closing a SCP session

Like in the SFTP subsystem, you don't handle the SSH channels directly. Instead, you open a "SCP session".

When you open your SCP session, you have to choose between read or write mode. You can't do both in the same session. So you specify either SSH_SCP_READ or SSH_SCP_WRITE as the second parameter of function ssh_scp_new().

Another important mode flag for opening your SCP session is SSH_SCP_RECURSIVE. When you use SSH_SCP_RECURSIVE, you declare that you are willing to emulate the behaviour of "scp -r" command in your program, no matter it is for reading or for writing.

Once your session is created, you initialize it with ssh_scp_init(). When you have finished transferring files, you terminate the SCP connection with ssh_scp_close(). Finally, you can dispose the SCP connection with ssh_scp_free().

The example below does the maintenance work to open a SCP connection for writing in recursive mode:

```
int scp_write(ssh_session session)
{
  ssh_scp scp;
  int rc;

  scp = ssh_scp_new
    (session, SSH_SCP_WRITE | SSH_SCP_RECURSIVE, ".");
  if (scp == NULL)
  {
    fprintf(stderr, "Error allocating scp session: %s\n",
            ssh_get_error(session));
    return SSH_ERROR;
  }

  rc = ssh_scp_init(scp);
  if (rc != SSH_OK)
  {
    fprintf(stderr, "Error initializing scp session: %s\n",
            ssh_get_error(session));
    ssh_scp_free(scp);
    return rc;
  }

  ...

  ssh_scp_close(scp);
  ssh_scp_free(scp);
  return SSH_OK;
}
```

The example below shows how to open a connection to read a single file:

```
int scp_read(ssh_session session)
{
  ssh_scp scp;
  int rc;

  scp = ssh_scp_new
    (session, SSH_SCP_READ, "helloworld/helloworld.txt");
  if (scp == NULL)
  {
    fprintf(stderr, "Error allocating scp session: %s\n",
            ssh_get_error(session));
    return SSH_ERROR;
  }

  rc = ssh_scp_init(scp);
  if (rc != SSH_OK)
  {
    fprintf(stderr, "Error initializing scp session: %s\n",
            ssh_get_error(session));
    ssh_scp_free(scp);
    return rc;
  }

  ...

  ssh_scp_close(scp);
  ssh_scp_free(scp);
  return SSH_OK;
}
```

Creating files and directories

You create directories with ssh_scp_push_directory(). In recursive mode, you are placed in this directory once it is created. If the directory already exists and if you are in recursive mode, you simply enter that directory.

Creating files is done in two steps. First, you prepare the writing with ssh_scp_push_file(). Then, you write the data with ssh_scp_write(). The length of the data to write must be identical between both function calls. There's no need to "open" nor "close" the file, this is done automatically on the remote end. If the file already exists, it is overwritten and truncated.

The following example creates a new directory named "helloworld/", then creates a file named "helloworld.txt" in that directory:

```
int scp_helloworld(ssh_session session, ssh_scp scp)
{
  int rc;
  const char *helloworld = "Hello, world!\n";
  int length = strlen(helloworld);

  rc = ssh_scp_push_directory(scp, "helloworld", S_IRWXU);
  if (rc != SSH_OK)
  {
    fprintf(stderr, "Can't create remote directory: %s\n",
            ssh_get_error(session));
    return rc;
  }

  rc = ssh_scp_push_file
    (scp, "helloworld.txt", length, S_IRUSR |  S_IWUSR);
  if (rc != SSH_OK)
  {
    fprintf(stderr, "Can't open remote file: %s\n",
            ssh_get_error(session));
    return rc;
```

```
  }

  rc = ssh_scp_write(scp, helloworld, length);
  if (rc != SSH_OK)
  {
    fprintf(stderr, "Can't write to remote file: %s\n",
            ssh_get_error(session));
    return rc;
  }

  return SSH_OK;
}
```

Copying full directory trees to the remote server

Let's say you want to copy the following tree of files to the remote site:

```
               +-- file1
       +-- B --+
       |       +-- file2
-- A --+
       |       +-- file3
       +-- C --+
               +-- file4
```

You would do it that way:

- open the session in recursive mode
- enter directory A
- enter its subdirectory B
- create file1 in B
- create file2 in B
- leave directory B
- enter subdirectory C
- create file3 in C
- create file4 in C
- leave directory C
- leave directory A

To leave a directory, call ssh_scp_leave_directory().

Reading files and directories

To receive files, you pull requests from the other side with ssh_scp_pull_request(). If this function returns SSH_SCP_REQUEST_NEWFILE, then you must get ready for the reception. You can get the size of the data to receive with ssh_scp_request_get_size() and allocate a buffer accordingly. When you are ready, you accept the request with ssh_scp_accept_request(), then read the data with ssh_scp_read().

The following example receives a single file. The name of the file to receive has been given earlier, when the scp session was opened:

```
int scp_receive(ssh_session session, ssh_scp scp)
{
  int rc;
  int size, mode;
  char *filename, *buffer;
```

```
  rc = ssh_scp_pull_request(scp);
  if (rc != SSH_SCP_REQUEST_NEWFILE)
  {
    fprintf(stderr, "Error receiving information about file: %s\n",
            ssh_get_error(session));
    return SSH_ERROR;
  }

  size = ssh_scp_request_get_size(scp);
  filename = strdup(ssh_scp_request_get_filename(scp));
  mode = ssh_scp_request_get_permissions(scp);
  printf("Receiving file %s, size %d, permissions 0%o\n",
          filename, size, mode);
  free(filename);

  buffer = malloc(size);
  if (buffer == NULL)
  {
    fprintf(stderr, "Memory allocation error\n");
    return SSH_ERROR;
  }

  ssh_scp_accept_request(scp);
  rc = ssh_scp_read(scp, buffer, size);
  if (rc == SSH_ERROR)
  {
    fprintf(stderr, "Error receiving file data: %s\n",
            ssh_get_error(session));
    free(buffer);
    return rc;
  }
  printf("Done\n");

  write(1, buffer, size);
  free(buffer);

  rc = ssh_scp_pull_request(scp);
  if (rc != SSH_SCP_REQUEST_EOF)
  {
    fprintf(stderr, "Unexpected request: %s\n",
            ssh_get_error(session));
    return SSH_ERROR;
  }

  return SSH_OK;
}
```

In this example, since we just requested a single file, we expect ssh_scp_request() to return SSH_SCP_REQUEST_NEWFILE first, then SSH_SCP_REQUEST_EOF. That's quite a naive approach; for example, the remote server might send a warning as well (return code SSH_SCP_REQUEST_WARNING) and the example would fail. A more comprehensive reception program would receive the requests in a loop and analyze them carefully until SSH_SCP_REQUEST_EOF has been received.

Receiving full directory trees from the remote server

If you opened the SCP session in recursive mode, the remote end will be telling you when to change directory.

In that case, when ssh_scp_pull_request() answers SSH_SCP_REQUEST_NEWDIRECTORY, you should make that local directory (if it does not exist yet) and enter it. When ssh_scp_pull_request() answers SSH_SCP_REQUEST_ENDDIRECTORY, you should leave the current directory.