Client-Server

A long-running daemon server process maintains the state of the repository. The repository is stored in a directory supplied to the server at command line when the server process is started. The file *contents* are stored in the repository using file names that are the MD5 hashes of their *contents*. In the directory given to the server to maintain the repository there may be a file called `.dedup` whose role is explained later. The server is responsible for correctly keeping track of which file name(s) correspond to which file contents.

The clients can connect to the server at any time. They issue commands sent as requests shown here and they receive responses from the server. Here `0x` is denoting the hexadecimal value of a single byte quantity.

List files (and response)

```
 client -> server
+-------- +
|  0x00  |
+-------- +
 server -> client
+.....  .....       +---------+----------+--+----------+--+----------  +----------+--+
| 0x01 |      N        | filename1 |\0| filename2 |\0|  ...   | filenameN |\0|
+.....  .....       +---------+----------+--+----------+--+----------  +----------+--+
```

`N` as a two byte quantity (most significant byte sent first) is the number of null-terminated (`\0`) filenames that follow the value `N`. `N` can be zero.

Upload file (and response)

```
client -> server
+--------  .....      +--+-----+..... +----+----                                    +
|  0x02  | filename |\0|        size       |  contents of the file (size bytes long) |
+--------  .....      +--+-----+..... +----+----                                    +
server -> client
+------+
| 0x03 |
+------+
```

`size` is a four byte (most significant byte sent first) quantity indicating the number of bytes of the file contents. `size` can be zero.

Delete file (and response)

```
client-> server
+--------                +--+
|  0x04  | filename |\0|
+--------                +--+
server-> client
+------+
| 0x05 |
+------+
```

the above is a successful deletion request.

Download file (and response)

```
client->server
+----------        +--+
|   0x06  | filename |\0|
+----------        +--+
server->client
+------+-----+ +----+----                                        +
| 0x07 |       size       |  contents of the file (size bytes long) |
+------+-----+ +----+----                                        +
```

where the meaning and structure of the fields is similar to the file uploading.

Quit (and response)

```
client->server
+       +
|  0x08  |
+       +
server->client
+------+
| 0x09 |
+------+
```

subsequently the server closes the connection to the server (and the same is expected for the client side to do as well)

All the above server responses prefixed by `0x01`, `0x03`, `0x05`, and `0x07` are for cases where the server has been able to provide a positive confirmation for the requested action. When the server determines it cannot issue a response as needed by the client, instead of sending back the corresponding `0x01`, `0x03`, `0x05`, or `0x07` response, it responds with an error indication which is prefixed by a byte with value `0xFF`. For example we may be trying to delete a file that does not exist. Then the servers can respond by:

```
server->client
+           +-----------+--+
|   0xFF  |  reason  |\0|
+           +-----------+--+
```

where `reason` is a string providing a humanly meaningful message about the error, like "file not found", etc. appropriate for the corresponding error. Note that the *Quit* request **cannot** return an error.

Server Behavior

The server (`ddupserver`) starts execution by calling it as:

```
ddupserver directory port
```

where `directory` is the path (relative or absolute) of the repository of files. When the server first executes called with a directory, the directory will be assumed to be empty. The `port` is the port to which the server will bind to listen to incoming client requests.

Sending a `SIGTERM` to the server should result in an orderly termination of the server. In orderly termination, the server saves it state in a file named `.dedup` in the repository directory. This file contains the mapping between stored files in the repository (that have MD5-derived file names) and the filenames with which they are known and its syntax follows simple XML syntax. For example

```
<?xml version="1.0"?>
<repository>
   <file>
      <hashname>3835946ed266331545915e22e13d5120</hashname>
      <knownas>foo.txt</knownas>
      <knownas>bar.txt</knownas>
   </file>
   <file>
      <hashname>b5884b223b0170166e05038919289334</hashname>
      <knownas>test.txt</knownas>
   </file>
</repository>
```

Naturally, each `<file>` element must have one `<hashname>` element and one or more `<knownas>` elements. When `ddupserver` is called with a directory that already contains a `.dedup` file, it uses the `.dedup` file to reinstate its state by appropriately populating its internal data structures.

An orderly termination of the server does not include any special message from the server informing the clients in any way that is shutting down. Simply the server, unilaterally, closes all the sockets to the clients before it exits.

Note: the following additional twist exists to the server behavior: once a file with a particular filename is uploaded to the server, its contents cannot be overwritten by another upload to the server that uses the same filename. A client has to first successfully request the removal/deletion of the filename before uploading a file with the same filename and (possibly) new contents. That is, the server does not support "overwrite" semantics.

Client Behavior

The client is invoked as follows:

```
ddupclient address port
```

where `address` is the IP address (can be local, i.e., 127.0.0.1) of the server and `port` is the port to which the server is listening. After the client connects to the server it sends the commands and handles the responses as described in the client-server protocol.

The user interface consists of line-at-a-time commands. The command syntax is a single letter (`l` for list, `u` for upload, `d` for download, `r` for delete, `q` for quitting). The `u`, `d`, and `r` commands are followed (in the same line) by one or more whitespace characters and then by the filename. Note that the client does not support pathnames for the files, only file names. It is assumed that a file to be uploaded should be searched in the current directory where `ddupclient` started. Equally, when a file is downloaded, it is downloaded in the directory in which `ddupclient` started.

The client reports the completion of the command from the response of the server as `OK` if it receives a server response `0x01`, `0x03`, `0x05`, `0x07`, or `0x09` from the server. The response of the `l` command as we have seen is a list of filenames, these are to reported, one filename per line, prefixed by the `OK+` before a final `OK` at the end of the list. The list can be empty, in which case, only the single ("last") `OK` is to be printed.

When a client receives `0x09`, it closes the connection to the server and exits.

If the client receives `0xFF` from the server then it reports `SERROR` followed in the same line by the `reason` string sent by the server. Note that neither the client nor the server are supposed to exit because of a `0xFF` response. If the client encounters an error on its side (e.g., trying to find a file in its local directory to upload), it does **not** send a request but reports a client-side error prefixed by `CERROR` followed by a short string explaining the reason for the error.

Here is an example session at the client (it assumes some files from other client(s) have already been uploaded):

```
l
OK+ first.pdf
OK+ somefile.txt
OK+ otherfile.txt
OK+ various.pdf
OK+ report.docx
OK
u myfile.doc
CERROR file not found in local directory
u myfile.txt
OK
u otherfile.txt

r somefile.txt
OK
d report.docx
OK
l
OK+ first.pdf OK+
otherfile.txt OK+
various.pdf OK+
report.docx OK+
myfile.txt
OK
r myfile.txt
OK
l
OK+ first.pdf
OK+ various.pdf
OK+ report.docx
OK+ otherfile.txt
OK
r myfile.txt
SERROR file not found
q
OK
(client exits at this point)
```

The example assumes that when starting the session, the local directory where `ddupclient` started contains a file `myfile.txt` and a file `otherfile.txt`. Note that the `u otherfile.txt` was an upload of a file by the client to the server but the server already has a file with this name. The client has pointlessly sent the contents of `otherfile.txt`, but the server dropped it because the existing `otherfile.txt` at the server has to be first removed.

Overwriting of files in the client local directory is *allowed*. That is if you download a file with a particular filename and a local file with the same filename exists, the local file is overwritten (replaced) by the newly downloaded one.