Project 4 — Get My Music (GMM)

David Brakman, Michael Dombrowski, Shiv Toolsidass

Protocol

Trust

The intended use of this software is to transfer files between hosts owned by the same individual. Therefore, all power and responsibility lies with the client, whose requests are trusted completely by the server.

Operations

1. List:
    - Spec:
        - Ask the server to return a list of files it currently has.
    - Implementation:
        1. Client sends a `listRequest` message to the server.
        2. Client receives a `listResponse` message from the server.
        3. Client prints message contents to console.
2. Diff:
    - Spec:
        - The client should show a "diff" of the files it has in comparison to the server - both files the server has that the client does not have, and files the client has and the server does not have.
        - "Diff" must actually crawl the directory structure and return all files in the current directory.
    - Notes:
        - The behavior of the Diff operation with regard to the following is unspecified:
            - Duplicate files (same contents, different name)
                - whose contents are only on the client
                - whose contents are only on the server
                - whose contents are on both the client and the server
                - Conflicting files (different contents, same name on the client and server)
            - In the spirit of the Unix `diff` command, our diff reports the changes that must occur to make the client "match" the server, i.e. the changes that a Sync operation will make.
        - A checksum is a robust means of determining whether or not two files contain the same content. Due to dimensionality reduction, a checksum cannot be perfectly accurate, but for this project, we consider two files with the same checksum to have the same content.
        - Consider two files with the same checksum to be "duplicate" files, and consider files for which no other file shares the same checksum to be "unique" files.
    - Implementation:
        1. Client sends a `listRequest` to the server and receives a `listResponse`.
        2. Client crawls its local directory, obtaining a list of files and their respective checksums.
        3. Client produces a `diffStruct`, enumerating:
            - Unique files that are present only on the client machine
            - Unique files that are present only on the server machine
            - Duplicate files that are present that are present only on the client machine
            - Duplicate files that are present only on the server machine

            - Duplicate files that are present on both the client and the server
        4. Client parses the `diffStruct` to determine and then print to the console the changes that would be made by a subsequent Sync operation.

3. Sync:

    - Spec:

        - Pulls to the local device all files from the server that the local device does not have and vice versa.
        - Students must develop a means of robustly determining whether or not two files, even if named differently, contain the same content (without actually sending the entire file).
        - The server must be multithreaded to handle multiple concurrent client requests (reads and writes).

    - Notes:

        - We consider a machine to "have" a given file iff a file in the directory has the same contents as the given file, even if the filenames associated with instances of those contents are different.
        - The behavior of the Sync operation with regard to files with the same name but different contents is unclear. If a server has file1 and a client has file1 and their contents differ, one possible course of action is to replace the older file with the newer file (unspecified behavior for exactly equal timestamps). We interpret the spec to mean that, after a Sync:
            - For each file on the client, at least one file on the server will have the same content.
            - For each file on the server, at least one file on the client will have the same content.

    - Implementation:

        1. Client sends a `listRequest` to the server and receives a `listResponse`.
        2. Client produces a `diffStruct` enumerating the differences between the local directory and the `listResponse`
        3. Client parses the `diffStruct` and builds the following:
            - a `pullRequest` asking the server to send a copy of:
                - each unique file present only on the server
                - the lexicographically first file of each set of duplicate files present only on the server
            - a `pushRequest` asking the server to receive a copy of:
                - each unique file present only on the client
                - the lexicographically first file of each set of duplicate files present only on the client
        4. Client sends the one `pullRequest` and the one `pushRequest` to the server
        5. Server receives a `pullRequest` and a `pushRequest`:
            - For each `(filename, checksum)` pair in the `pullRequest`, if a file exists on the server with the same filename and checksum, add to a `pullResponse` a corresponding tuple of `(filename, checksum, data)`. After processing all files listed in the `pullRequest`, send the `pullResponse` to the client.
            - For each `(filename, checksum, data)` pair in the `pushRequest`, the file with the given filename will be (over)written with `data`. No checksums are inspected for a `pushRequest`, because the server must trust a `pushRequest` as authoritative. The only reason for a functioning client to send a `pushRequest` for a file whose checksum is already present on the server is in the rare case that different contents map to the same checksum and an advanced user is forcing an overwrite. The server then sends a `pushResponse` containing a list of `(filename, checksum)` tuples of every file in the request that was written to the server (which must be all of them).
        6. Client receives a `pushResponse` and a `pullResponse`. If either response contains a shorter list of files than was sent in the corresponding request, print that the Sync failed. Otherwise, print to the console a list of filenames whose data was written to each host during the Sync.

4. Leave:

    - Spec:

        - The client should end its session with the server and take care of any open connections.
        - The server must ... store/retrieve historical information about each client in a file.

    - Implementation:

        1. Server receives a `leave` message from the client.
        2. Server closes the socket and appends ": Client closed connection" to a log file.

Messages

Messages exchanged over the network will be strings adhering to a limited subset of JSON (RFC 7159). We chose this higher-level message format, instead of a more concise custom format like the one in the specification for Project 3, with the goal of simplifying development, a more pressing concern than minimizing bandwidth usage. Instead of using bit manipulation, we can use a map data structure to build and parse messages according to human-readable keys.

All messages in our protocol will share a common envelope:

```
{
  "version": <Integer>,
  "type": <String>,
}
```

- Version: An integer i corresponding to the version of the message format being sent. If a message format breaks compatibility with existing implementations, its version number should be incremented so that communicating parties can recognize the issue and either adapt or abort. The development version articulated here is version 1.
- Type: the type of the message being sent, identified by one of the following strings: "listRequest", "listResponse", "pullRequest", "pullResponse", or "leave". This type info informs the recipient what key-value pairs can be expected in the rest of the message. Notably, there are no "sync" or "diff" messages as these are entirely client-side.
  - **listRequest:** no additional information. Example:

    ```
    {
      "version": 1,
      "type": "listRequest",
    }
    ```

  - **listResponse:** response info consisting of a list of (filename, checksum) tuples. Example:

    ```
    {
      "version": 1,
      "type": "listResponse",
      "response": [
        {
          "filename": "file1",
          "checksum": "aaabaaajss",
        },
        {
          "filename": "file2",
          "checksum": "wg2gnxgrrr",
        }
      ]
    }
    ```

  - **pullRequest:** request info consisting of a list of (filename, checksum) tuples. Example:

    ```
    {
      "version": 1,
      "type": "pullRequest",
      "request": [
        {
          "filename": "file1",
          "checksum": "aaabaaajss",
        },
    ```

```
    {
      "filename": "file2",
      "checksum": "wg2gnxgrrr",
    }
  ]
}
```

- **pullResponse:** response info consisting of a list of (filename, checksum, data) tuples. To be sent as a string, file data will be encoded in base64. If a pullRequest includes invalid (filename, checksum) pairs that do not exist on the server, the pullResponse will send only the files corresponding to valid (filename, checksum) pairs. Example:

```
{
  "version": 1,
  "type": "pullResponse",
  "response": [
    {
      "filename": "file1",
      "checksum": "aaabaaajss",
      "data": "bGFsYWxhIGhhcHB5IHN0cmluZw=="
    },
    {
      "filename": "file2",
      "checksum": "wg2gnxgrrr",
      "data": "z/rt/gcAAAEDAACAAgAAABAAAACwBQAAhQAgAAAAAAAZAAAASAAA"
    }
  ]
}
```

- **pushRequest:** request info consisting of a list of (filename, checksum, data) tuples. To be sent as a string, file data will be encoded in base64. Example:

```
{
  "version": 1,
  "type": "pushRequest",
  "request": [
    {
      "filename": "file3",
      "checksum": "AATTVVV",
      "data": "bGFsYWxhIGhhcHB5IHN0cmluZw=="
    },
    {
      "filename": "file4",
      "checksum": "ABBBBBBK",
      "data": "z/rt/gcAAAEDAACAAgAAABAAAACwBQAAhQAgAAAAAAAZAAAASAAA"
    }
  ]
}
```

- **pushResponse:** response info consisting of a list of (filename, checksum) pairs of the files written to the server in response to a given `pushRequest`. Example:

```
{
  "version": 1,
  "type": "pushResponse",
  "response": [
    {
      "filename": "file3",
      "checksum": "AATTVVV",
    },
    {
      "filename": "file4",
      "checksum": "ABBBBBBK",
    }
  ]
}
```

- **leave:** no additional information. Example:

```
{
  "version": 1,
  "type": "leave",
}
```

Client

Server

Multithreading