YourTunes File System

Sean Collins, Tim Farley, Robert Hensey

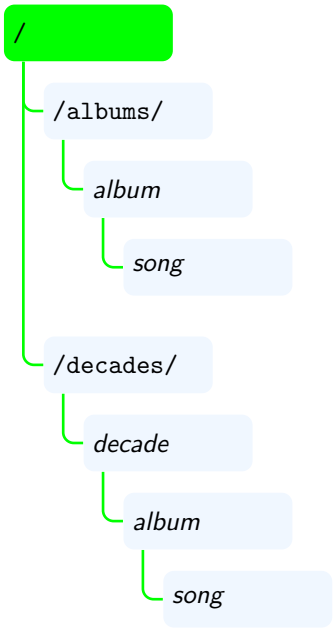
CS 3210

1 Introduction

The **YourTunes File System** is a FUSE-based implementation of a filesystem tailored towards music files. It is built on top of a remote backend server, which stores file data and metadata on a database server with a RESTful web server frontend. The client is responsible for transparently querying the remote server for resident files and constructing a metadata-based abstraction based on the parsed server response.

1.1 Directory structure

The directory structure on the local filesystem is based on audio metadata - specifically, the `Album`, `Title`, `Track`, and `Year` ID3 fields.



Each song is placed as a leaf in two separate directory trees: one based on albums directly, and one based on albums categorized by the decade that they were released in (parsed from the `Year` field). In the case of missing metadata, a song is placed in a "Unknown" folder for both cases (`/albums/Unknown/`*file*, `/decades/Unknown/Unknown/`*file*). Files with no metadata are also put into these buckets.

Given the same metadata, the file pointers through both directory hierarchies are pointers to the same file data.

1.2 Example

Given an audio file with the following metadata:

Track	4
Title	Jesus, Take the Wheel
Artist	Carrie Underwood
Album	*Some Hearts*
Year	2005

The filesystem will present the following abstraction:

```
/
├── albums/
│   └── Some Hearts/
│       └── 4-Jesus, Take the Wheel
└── decades/
    └── 2000s/
        └── Some Hearts/
            └── 4-Jesus, Take the Wheel
```

2 Client

2.1 Setup

The following packages are required: `libfuse-dev` `pkg-config`, `mp3info`, `curl`, `jq`. Assuming a Debian-based system, these can be installed by running `./install-deps.sh` or running the following command manually:

```
sudo apt-get install -y libfuse-dev pkg-config mp3info curl jq
```

Once the dependencies are installed, run `make` from the project root to build the client. Finally, run `./yourtuneslib <mountpoint>` to mount the filesystem to a local directory, with `<mountpoint>` being the path to an existing directory.

3 Server

The backing server serves as the file and metadata store for the filesystem. It provides an interface to list, add, and remove files as well as query for metadata.

3.1 API

3.1.1 `GET /ls`

Parameters: *None.*

Returns a recursive listing of all files on the filesystem.

3.1.2 `GET /get_file/`*filename*

Parameters: `filename` The name of the file (as output by `/ls` above).

Serves a raw file given by the filename through HTTP.

3.1.3 `GET /delete_file/`*filename*

Parameters: `filename` The name of the file (as output by `/ls` above).

Deletes a file given by the filename from the server.

3.1.4 `POST /upload`

Parameters: `file_data` The raw file data.

Uploads a file to the file store.

3.2 Setup

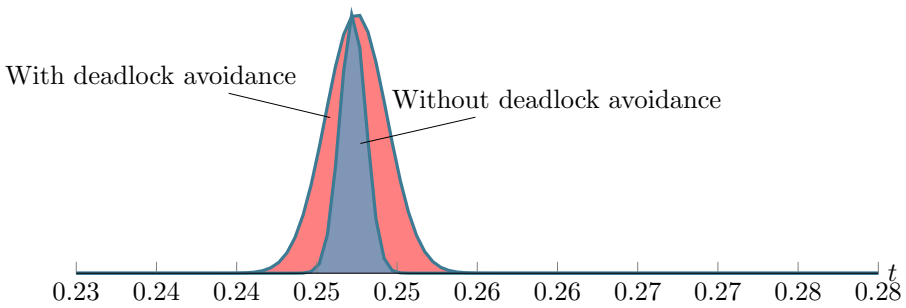
Run `./install.sh` in the `server` directory. You will need sudo privileges as well as a root MySQL database password, and will be prompted for these when required. If MySQL is not installed, it will be installed for you, with the root password being set to `team14`.

Once the script completes, start the server by running `python3 server.py`.

4 Performance benchmarks

Tests were performed with the included `perf-test-no-fuse.sh` and `perf-test-with-fuse.sh` test programs, which measure nanosecond-precision performance copying a test file from a local source to destination. For the purposes of comparison, the first test copies the file to a local folder, and the second test copies the file to a local FUSE mountpoint (a remote destination).

Run them with `./perf-test-no-fuse.sh` and `./perf-test-with-fuse.sh`, respectively.



	Min	Max	Mean	Std. Div.
Copy to local folder	TODO s	TODO s	**TODO s**	**TODO s**
Copy to FUSE mountpoint	TODO s	TODO s	**TODO s**	**TODO s**
Distance	+TODO s	+TODO s	**+TODO s**	**+TODO s**

4.1 Analysis