MPEG Sorter Technical Documentation

A Python utility that identifies and sorts media files based on their actual file signatures rather than extensions. Accurately separates MP3 and MP4 files into appropriate directories, correcting mislabeled extensions in the process.

Implementation Details

File Structure

```
mpeg-sorter/            - Module directory
  mpeg_sorter.py        - Main Python script
tests/                  - Test directory
  conftest.py           - Pytest configuration
  test_mpeg_sorter.py   - Main test script
  test_mpeg_sorter_pytest.py - Pytest-compatible test script
  data/                 - Test data directory
```

Dependencies

The implementation relies solely on Python standard library modules: - `os`: For file system operations and CPU core detection - `shutil`: For moving files - `argparse`: For command-line argument parsing - `pathlib`: For path manipulation - `asyncio`: For asynchronous execution framework - `concurrent.futures`: For parallel processing with ThreadPoolExecutor - `time`: For performance measurement

Core Components

1. **File Signature Detection (`get_file_signature`):**
    - Reads the first 12 bytes of each file
    - Compares against known MP3 and MP4 signatures
    - Returns the identified file type ('mp3', 'mp4', or 'unknown')
2. **File Sorting Logic (`sort_files` and `sort_files_async`):**
    - Creates appropriate subdirectories
    - Processes files in the source directory (non-recursively)
    - Uses asynchronous parallel processing with ThreadPoolExecutor
    - Identifies file types using signatures
    - Handles extension mismatches
    - Moves files to destination folders
    - Resolves filename conflicts
    - Provides real-time progress updates
3. **Command-line Interface (`main`):**
    - Parses command-line arguments

- Validates inputs
- Provides help information
- Includes worker count configuration option
- Offers sequential mode for performance benchmarking
- Creates unknown folder by default (can be disabled with `--no-unknown`)
- Calls the appropriate sorting function based on chosen mode

Signature Detection

The script uses the following signatures for detection:

MP3 Signatures

- `\xFF\xFB`: MPEG-1 Layer 3
- `\xFF\xF3`: MPEG-2 Layer 3
- `\xFF\xF2`: MPEG-2.5 Layer 3
- `\x49\x44\x33`: ID3 tag (common in MP3)

MP4 Signatures

- `\x00\x00\x00\x18\x66\x74\x79\x70`: ISO Base Media file (MPEG-4)
- `\x00\x00\x00\x20\x66\x74\x79\x70`: ISO Base Media file (MPEG-4)
- `\x66\x74\x79\x70\x4D\x53\x4E\x56`: MPEG-4 video
- `\x66\x74\x79\x70\x69\x73\x6F\x6D`: ISO Base Media file (MPEG-4)

Additional check for 'ftyp' after the initial 4 bytes is also performed for MP4 container formats.

Analysis of Strengths and Weaknesses

Strengths

1. **Signature-based Detection**
   - More reliable than extension-based sorting
   - Can identify mislabeled files with high accuracy
   - Minimal false positives for supported formats
2. **Efficient Implementation**
   - Minimal memory footprint (reads only first 12 bytes of each file)
   - No external dependencies required
   - Files are moved rather than copied, saving disk space
3. **User Experience**
   - Clear feedback for each operation
   - Handles edge cases like duplicate filenames
   - Intuitive command-line interface
4. **Safety Features**
   - Validates source directory existence

- Creates subdirectories as needed
- Exception handling for file operations

Weaknesses

1. **Limited Format Support**
   - Currently only handles MP3 and MP4 formats
   - No support for other audio formats (WAV, FLAC, OGG, etc.)
   - No support for other video formats (AVI, MKV, WebM, etc.)
2. **Basic Detection Method**
   - Relies on a small set of signatures for each format
   - May miss some valid files with unusual headers
   - No advanced analysis of file contents beyond headers
3. **No Recursive Processing**
   - Only processes files in the top level of the source directory
   - Ignores files in subdirectories
4. **Asynchronous Parallel Processing**
   - Uses ThreadPoolExecutor for concurrent file processing
   - Dynamically scales to available CPU cores for optimal performance
   - Includes progress indication for monitoring large operations
   - Significantly improves processing speed for large collections

Future Improvements

1. **Expanded Format Support**
   - Add support for additional audio formats (FLAC, WAV, AAC, OGG)
   - Add support for additional video formats (MKV, AVI, WebM, FLV)
   - Create a more extensible signature database
2. **Enhanced Detection**
   - Implement more comprehensive signature detection
   - Add secondary verification methods for ambiguous files
   - Consider using content-based analysis for better accuracy
3. **Performance Enhancements**
   - Add multi-threading support for faster processing
   - Implement progress bars for large operations
   - Add options for batch processing
4. **Additional Features**
   - Add recursive directory processing
   - Add support for custom destination directories
   - Implement dry-run mode to preview changes
   - Add metadata extraction for better organization
   - Create file format reports and statistics
5. **User Interface Improvements**
   - Add a simple GUI option
   - Implement color-coded console output
   - Add interactive mode for uncertain file types

6. **Testing and Validation**
    - Add unit tests for reliability
    - Create test files for various edge cases
    - Implement validation for sorted files

Implementation Considerations

For future development, consider:

1. **Modularity**: Separate concerns into distinct modules for better maintainability:
    - Signature detection
    - File operations
    - CLI handling
    - Logging/reporting
2. **Configuration**: Add a configuration file for customizable settings:
    - Custom signature definitions
    - Default behavior options
    - Custom destination paths
3. **Performance**: For larger collections, consider:
    - Batch processing to reduce system calls
    - Memory-mapped file access for faster signature detection
    - Worker pools for parallel processing
4. **Integration**: Consider adding hooks for:
    - Media library management systems
    - Backup solutions
    - Automated workflows

By addressing these improvements, MPEG Sorter could evolve into a more comprehensive media management utility suitable for various use cases, from personal collections to professional media libraries.

Testing Framework

The testing framework consists of the following components:

1. **Main Test Script (`test_mpeg_sorter.py`)**
    - Contains the main test logic with command-line interface
    - Handles file creation, validation, and cleanup
    - Supports both direct function testing and command-line script testing
    - Automatically adapts to the project structure
    - Best for quick testing and performance benchmarking
2. **Self-contained Pytest Test Script (`test_mpeg_sorter_pytest.py`)**
    - Fully independent test file specifically designed for pytest
    - Contains all necessary test classes and helpers with no external dependencies

    - Provides individual test functions for granular test reporting
    - Includes specific tests for each processing mode
    - Automatically skips tests that aren't applicable to the current environment
    - Best for CI/CD integration and detailed test reporting
3. **Pytest Configuration (`conftest.py`)**
    - Sets up proper import paths for all project modules
    - Provides shared fixtures for all tests
    - Enables command-line options for test customization
    - Works with multiple module directories in the project structure
4. **Test Data Management**
    - Automatically creates sample files with correct signatures when needed
    - Uses a separate data directory for test file storage
    - Creates a temporary test directory for each test run
    - Restores the test environment to its original state after testing

Testing

Overview

The MPEG Sorter includes a comprehensive test suite that validates its functionality and provides performance benchmarking between sequential and parallel processing modes.

Test Setup

The test suite automatically:

1. Sets up a test environment with sample media files
2. Runs the sorter in both sequential and parallel modes
3. Validates that files are correctly sorted and renamed
4. Restores the original file structure for easy re-testing
5. Reports performance metrics and speedup comparisons

Test Data

The test script automatically generates the following test files if they don't already exist:

Filename	Actual Type	Description
video1.mp4	MP4	Correctly labeled video file
video2_as_audio.mp3	MP4	Video file with incorrect .mp3 extension
audio1.mp3	MP3	Correctly labeled audio file
audio2_as_video.mp4	MP3	Audio file with incorrect .mp4 extension

Filename	Actual Type	Description
unknown.bin	Unknown	File with unknown signature

Running the Tests

```
# Run the test script directly (from project root)
python tests/test_mpeg_sorter.py

# For detailed output
python tests/test_mpeg_sorter.py --verbose

# Force command-line script testing
python tests/test_mpeg_sorter.py --command-line
```

The test script will automatically: 1. Locate the module in the project structure 2. Create and initialize test files if needed 3. Run the tests in both sequential and parallel modes 4. Clean up after completion

Integration with pytest The project includes a dedicated pytest-compatible test file that works reliably in all environments:

```
# Run with the pytest-specific test file (recommended)
pytest tests/test_mpeg_sorter_pytest.py -v

# Run specific tests only
pytest tests/test_mpeg_sorter_pytest.py::test_command_line_sequential -v

# Run with test selection by keyword
pytest tests/test_mpeg_sorter_pytest.py -v -k "command"

# Run with additional output detail
pytest tests/test_mpeg_sorter_pytest.py -vv
```

The pytest-compatible test file provides several advantages: - Self-contained implementation with no external dependencies - Individual test functions that appear in pytest reports - Automatic skipping of tests that aren't applicable - Detailed output of test results with pass/fail status for each test - Better integration with CI/CD pipelines and test reporting tools

The `conftest.py` file enhances pytest functionality by: - Setting up Python import paths for all project modules - Enabling module imports regardless of the directory structure - Providing command-line options for customizing test behavior - Allowing tests for different modules to coexist in the test suite

Test Results

The test will output:

1. Pass/fail status for each test mode
2. Performance timing for each mode
3. Performance speedup comparison between sequential and parallel modes

Example output:

```
============================================================
RUNNING MPEG SORTER TESTS
============================================================

[TEST] Running in sequential mode...
[PASS] Sequential mode sorting completed successfully

[TEST] Running in parallel mode...
[PASS] Parallel mode sorting completed successfully

============================================================
TEST SUMMARY
============================================================
Sequential Mode: PASSED (0.1234 seconds)
Parallel Mode: PASSED (0.0345 seconds)

Performance Speedup: 3.58x faster in parallel mode
```

Validation Checks

The test suite verifies:

1. Correct directory creation (audio, video, unknown)
2. Proper file sorting based on signature detection
3. Correct extension renaming for mislabeled files
4. Complete restoration of the test environment after testing

Test Cleanup

The test automatically cleans up after itself by:

1. Moving all files back to their original locations
2. Restoring original filenames
3. Removing created directories
4. Cleaning up the temporary test directory

This ensures you can run the tests repeatedly without manual cleanup between runs.