MODradio - dokumentacja projektu

Grzegorz Koperwas

Wydział Matematyki Stosowanej, Politechnika Śląska

16 stycznia 2022, Gliwice

1. Temat projektu:

Celem powstałego programu jest strumieniowanie piosenek z `trackerów` poprzez *Http Live Streaming*[1] w celu łatwego odsłuchu na urządzeniach mobilnych.

W tym celu musi on dynamicznie łączyć kolejne pliki w jeden strumień, bez wcześniejszego wczytania ich wszystkich[2] do pamięci.

Opis problemu:

Archiwa strony `modarchive.org` są udostępniane w następującej formie:

1. Piosenki znajdują się w drzewie folderów rozróżniającym je ze względu na format, artystę czy rok dodania.

   Jakiekolwiek informacje zawarte w strukturze folderów mają być ignorowane, odtwarzamy piosenki w prawdziwie losowy sposób.

   Odtwarzacz VLC może uzyskiwać dostęp po protokole SMB do serwera z plikami, lecz nie odtwarza ich losowo.

2. Każda piosenka jest skompresowana jako archiwum `ZIP`.

3. Piosenki są przechowywane jako *moduły trackerów*, gdzie zamiast danych `PCM` przechowywane są w formie sampli i informacji jak je odtwarzać. Wynika to z architektury komputerów *Amiga*, gdzie ten rodzaj muzyki powstał.

Zatem program musi:

1. Przyjmować pliki audio w formie pozwalającej na utrzymywanie *bufora* następnych plików audio.

2. Przyjmować pliki, lub ścieżki do nich ze źródła łatwo dostępnego dla jakiegoś skryptu. Na przykład przez `stdin`.

3. Spełniać wymagania do łatwego zahostowania na moim klastrze *Docker Swarm*, brak GUI itp.

Ważne rzeczy na które warto zwrócić uwagę:

1. Działanie programu było weryfikowane w środowisku:

   - System zgodny z `POSIX`, tutaj `Arch Linux`
   - `OpenJDK Runtime Environment (build 11.0.13+8)`
   - `Apache Maven 3.8.4`

2. Program używa natywnych bibliotek, przez to lista wspieranych platform ogranicza się do poniższych kombinacji architektur i systemów operacyjnych:

[1] Dalej będę korzystał ze skrótu **HLS**

[2] Ilość tych plików wynosi 29 tysięcy, rozmiar około 50 gigabajtów w formie skompresowanej

- Android (arm, arm64, x86, x86_64)
- iOS (arm, x86_64[3])
- linux (armhf, arm64, ppc64le, x86, x86_64)
- macOS (arm64, x86_64)
- Windows (x86, x86_64)

3. Dokumentacja jest kompilowana korzystając z narzędzia `xelatex`, jednak diagramy są generowane skryptem `dot2tex`. Plik `./Makefile` zawiera reguły dla programu `make`, które kompilują dokumentację po wydaniu polecenia `make docs/doc.pdf`.

2. Opis pobieranych danych przez program:

Program pobiera przez standardowe wejście ścieżki do kolejnych plików audio. Pliki te są **usuwane** po zakończeniu ich strumieniowania.

Program po napotkaniu pliku, którego nie da się otworzyć lub zdekodować pomija dany plik.

Program stara się otwierać dwa pliki naraz. Plik, który jest właśnie strumieniowany, oraz plik, który jest następny w kolejce.

Program wspiera wiele formatów wejściowych, ich dokładna lista zależy od flag użytych do kompilacji biblioteki ffmpeg. Jeżeli plik wejściowy zawiera więcej niż jeden strumień[4], to program wybiera pierwszy strumień audio.

3. Opis otrzymanych rezultatów

Wydruk z programu

Program wypisuje do konsoli logi diagnostyczne, za przechowywanie ich w plikach odpowiada **systemd** lub **docker**.

Przykładowe logi znajdują się na załączniku 1. Logi w załączniku składają się z paru części:

- Logi ze znakami < oraz >, - Logi informujące jakie obiekty są tworzone przez program. Przykładowo:
  - `<Encoder for aac>` - Stworzono obiekt kodera formatu `AAC`, zawsze wyświetla się na początku programu.
  - `<Reader for /path/to/file>` - Stworzono obiekt demuxera, który czyta zawartość pliku. Jeżeli dany plik zawiera parę strumieni wideo lub audio, program wybiera pierwszy strumień audio. Powinien wspierać większość formatów[5], w tym na przykład `mp4`.

[3] Używane dla emulatora systemu iOS, same urządzenia wykorzystują tylko arm

[4] Na przykład plik wideo będzie zawierał zwykle jedną ścieżkę wideo, jedną lub więcej audio oraz nawet kilkanaście ścieżek napisów, czasami nawet miniaturkę jako jednoklatkowy strumień MJPEG.

[5] Pewnie nawet zasoby sieciowe, zależy od wersji biblioteki `libav`.

  - `<Decoder for $codec>` - Stworzono obiekt dekodera kompresji, program wspiera wiele różnych kodeków. Dokładna lista zależy od opcji użytych do skompilowania pakietu `org.bytedeco:ffmpeg-platform`.
  - `<Resampler from $foo to $bar>` - Stworzono obiekt resamplera, który normalizuje częstotliwość próbkowania oraz zapis bitowy.

- Logi z znakami [oraz][6], - Logi generowane przez bibliotekę `libav` - są zwykle w formie:

  [$źródło @ adres źródła] $wiadomość

  Zwykle są to logi o statusie muxera `HLS`, lecz w przypadku złego pliku wejściowego zawierają one dodatkowe informacje o błędzie. Program `ffmpeg`, który jest frontendem do biblioteki `libav` generuje te same logi, więc jego dokumentacja pomoże w diagnozowaniu problemu.

- Reszta:

  Część logów w przykładzie pochodzi od skryptu realizującego przykładowe wykorzystanie programu. Jest on dołączony do kodu jako `./feeder.sh`.

Pliki tworzone przez program:

Program tworzy w aktualnym katalogu strumień w formacie `HLS` jako pliki `stream$x.ts` oraz plik „spis” `stream.m3u8`. Te pliki powinny być hostowane przez serwer HTTP, jako pliki statyczne. Programy takie jak *VLC Media Player*, *ffplay*, *safari* czy przeglądarki internetowe na systemie android odtworzą je bez problemu, nawet z dysku. Dla przeglądarek na komputerach PC trzeba dostarczyć demuxer w *JavaScript'cie*, co nie jest przedmiotem tego projektu.

[6] te kolorowe

```
[mpegts @ 0x7f0600355480] frame size not set
<Encoder for aac>
extracted /tmp/modfiles/lazertrack_heaven_2.mod
[aac @ 0x7f05f4005240] Estimating duration from bitrate, this may be inaccurate
<Reader for /tmp/modfiles/lazertrack_heaven_2.mod.aac>
<Decoder for aac>
<Resampler from 44100hz to 44100hz>
[hls @ 0x7f060034de00] Opening 'stream0.ts' for writing
[hls @ 0x7f060034de00] Opening 'stream.m3u8.tmp' for writing
extracted /tmp/modfiles/the_hardliner_-_whoronzon_gohonzon.xm
[aac @ 0x7f05ec001100] Estimating duration from bitrate, this may be inaccurate
<Reader for /tmp/modfiles/the_hardliner_-_whoronzon_gohonzon.xm.aac>
extracted /tmp/modfiles/the_savannus_never_never.669
[hls @ 0x7f060034de00] Opening 'stream1.ts' for writing
[hls @ 0x7f060034de00] Opening 'stream.m3u8.tmp' for writing
[hls @ 0x7f060034de00] Opening 'stream2.ts' for writing
[hls @ 0x7f060034de00] Opening 'stream.m3u8.tmp' for writing
extracted /tmp/modfiles/gustavo6046_-_trulix.it
extracted /tmp/modfiles/jabdah-cover.xm
extracted /tmp/modfiles/jason_ee-futurefuckballs2010_cover.it
extracted /tmp/modfiles/owcfullfrontal.it
[hls @ 0x7f060034de00] Opening 'stream3.ts' for writing
[hls @ 0x7f060034de00] Opening 'stream.m3u8.tmp' for writing
extracted /tmp/modfiles/skyline_-_boners.it
extracted /tmp/modfiles/ko0x_-_galaxy_guppy.it
[hls @ 0x7f060034de00] Opening 'stream4.ts' for writing
[hls @ 0x7f060034de00] Opening 'stream.m3u8.tmp' for writing
extracted /tmp/modfiles/pasyada_alex_-_decil.xm
extracted /tmp/modfiles/badboyremixhypnosis.mod
[hls @ 0x7f060034de00] Opening 'stream5.ts' for writing
[hls @ 0x7f060034de00] Opening 'stream.m3u8.tmp' for writing
Waiting for modradio to pickup data
[hls @ 0x7f060034de00] Opening 'stream6.ts' for writing
[hls @ 0x7f060034de00] Opening 'stream.m3u8.tmp' for writing
[hls @ 0x7f060034de00] Opening 'stream7.ts' for writing
[hls @ 0x7f060034de00] Opening 'stream.m3u8.tmp' for writing
...
```

Załącznik 1: Przykładowe logi z programu.

4. Zastosowane algorytmy:

Obieg danych jest przedstawiony na załączniku 2.

W programie został zastosowany mechanizm asynchroniczności poprzez klasę standardową `Future` oraz `ReaderAsyncFactory`, proces ten jest pokazany w załączniku 3.

Ogólny schemat blokowy jest pokazany w załączniku 4.

Opis poszczególnych klas:

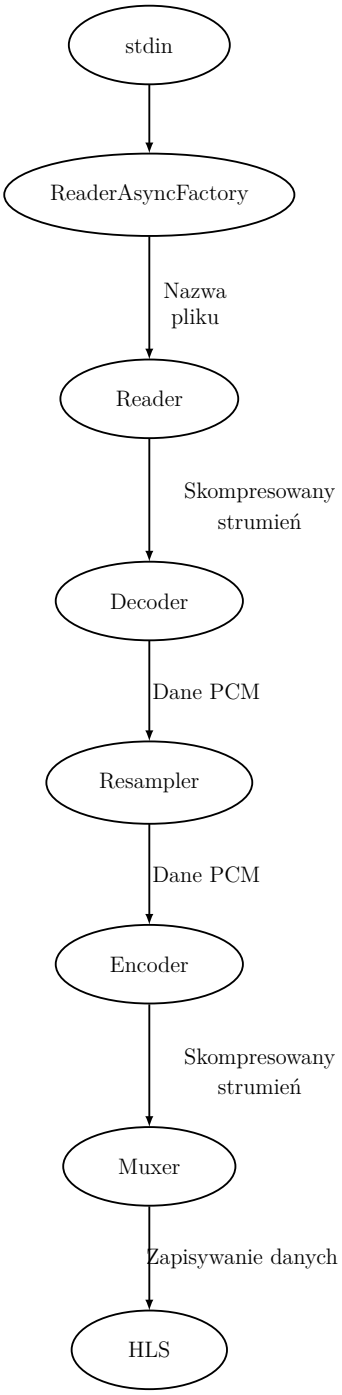
1. `App`

Klasa zawierająca główną logikę programu w metodzie `main`.

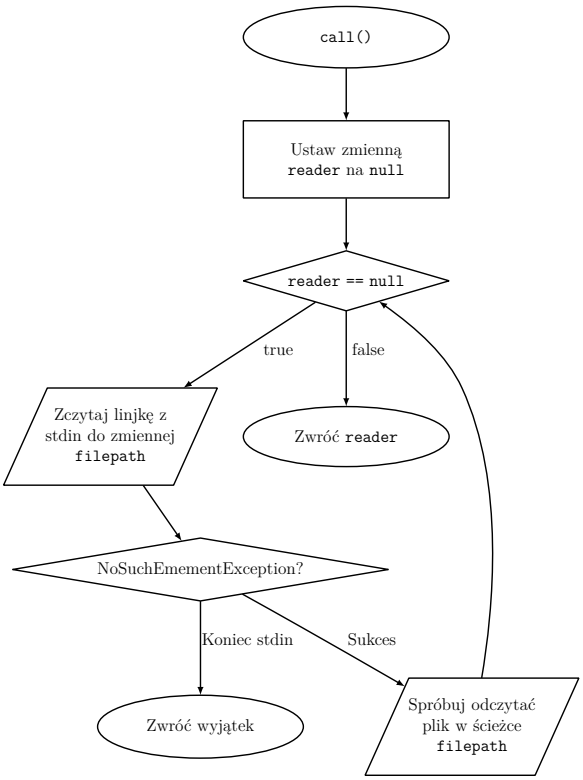
2. `Reader`

Klasa odpowiedzialna jest za:

- Otwieranie demuxera dla zadanego pliku.
- Wybieranie strumienia audio w pliku.



Załącznik 2: Pipeline danych



Załącznik 3: Schemat blokowy działania klasy `ReaderAsyncFactory`

- Zwracanie kolejnych pakietów skompresowanego strumienia.

Jest ona zrealizowana głównie jako obudowanie struktury `AVFormatContext` z biblioteki `libav`.

3. `ReaderAsyncFactory`

Klasa implementuje `Callable<Reader>`. Z tego powodu bardziej zachowuje się jako funkcja, której działanie znajduje się na diagramie w załączniku 3. Zrealizowana jest w ten sposób by ewentualny proces „zgadywania" czy dany plik czymś użytecznym nie wpływał na główny wątek.

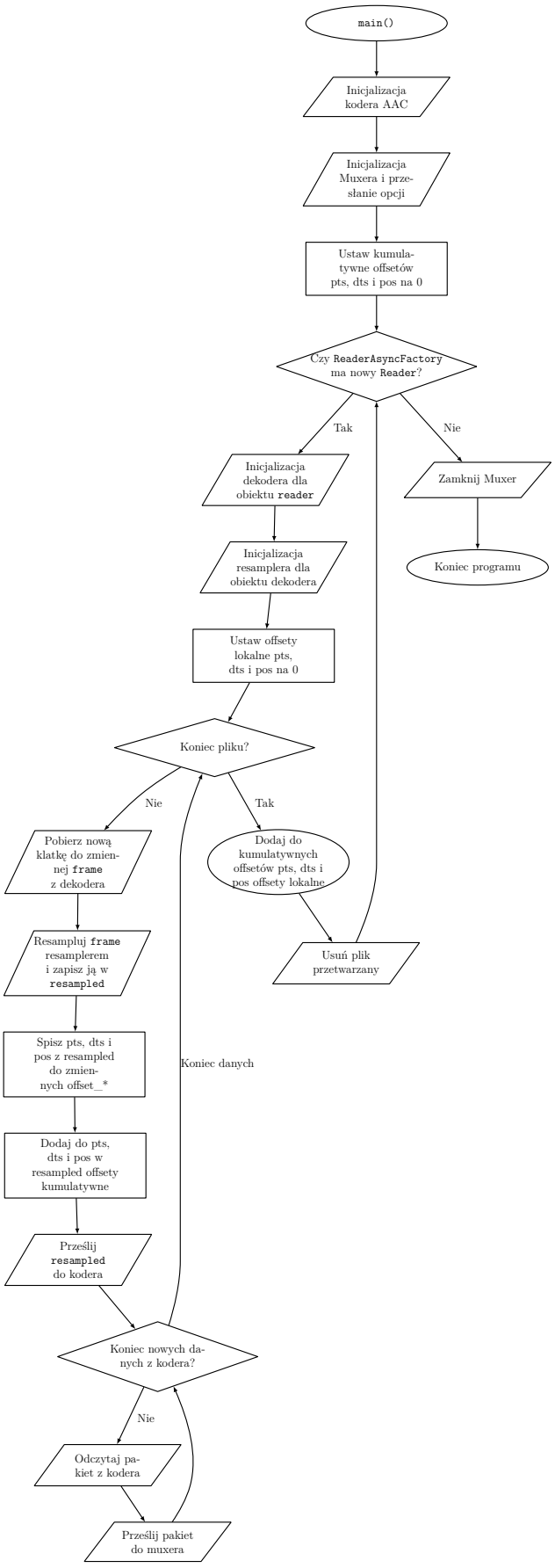
4. `Decoder`

Klasa opakowuje obiekty klasy `Reader`, w celu zdekodowania informacji. Odpowiedzialna jest za:

- Stworzenie kontekstu kodeka `AVCodecContext`.
- Otworzenie samego kodeka w bibliotece `libav`.
- Zwracanie kolejnych zdekodowanych klatek.

5. `Encoder`

Klasa zarządza procesem kodowania klatek. Odpowiedzialna jest za:



Załącznik 4: Ogólny schemat działania programu.

- Tworzenie kontekstu kodeka `AVCodecContext` z zadanymi parametrami:
  - Rodzaj kodeka (używany jest tylko AAC)
  - Bitrate strumienia
  - Format bitowy próbek, który musi być wspierany przez kodek.
  - Częstotliwość próbkowania
  - Układ kanałów audio
- Przekazywanie klatek do zakodowania do kodeka.
- Odbieranie zakodowanych pakietów, oraz informowanie kiedy takowe się już skończyły.

6. `Resampler`

Klasa konwertuje różne parametry strumieni dekodowanych przez obiekty klasy `Decoder`, tak by mogły one dzielić jeden obiekt klasy `Encoder`. Odpowiada za:

- Inicjalizację kontekstu resamplera `SwrContext`, w oparciu o parametry wejściowego oraz wyjściowego `AVCodecContext`.
- Przepisywanie `pts`, `dts` oraz `pos` między klatkami.

7. `Muxer`

Klasa odpowiedzialna jest za:

- Otwieranie muxera dla zadanego pliku.
- Tworzenie strumienia audio w oparciu o `AVCodecContext` `Encoder`'a.
- Zapisywanie kolejnych pakietów w czasie rzeczywistym.

5. Testy na poprawność działania programu:

Poprawność działania była sprawdzana poprzez odsłuch strumieni generowanych przez program oraz hostowanie ich serwerem `http` z biblioteki standardowej języka *Python*, poprzez komendę `python -m http.server`.

6. Wnioski:

Znane błędy

Program generuje strumień w tempie leciutko zawyżonym. Podczas dłuższego odsłuch może wystąpić konieczność przewinięcia do przodu strumienia, bo starsze części mogą już nie być dostępne. Wynika to z niejasności dokumentacji `libav` na temat tego, w jakim formacie są dane synchronizujące zawarte w klatkach strumienia. Przykładowo pliki w formacie `mp3` oraz `aac` mają kompletnie różne wartości.

Jednym z możliwych rozwiązań tego problemu byłoby manualne obliczanie długości poszczególnych klatek w resamplerze, na podstawie ilość próbek (sampli), częstotliwość próbkowania oraz ilość kanałów.

Innym błędem, raczej związanym z poprzednim[7], jest błąd przy odczycie plików w formacie `mp3`, program otwiera je poprawnie, resampluje z 48000hz do 41000hz, koduje do mp3, ale efekt wyjściowy nie przypomina zbytnio pliku wejściowego.

Inne:

Po stworzeniu tego programu lepiej rozumiem architekturę biblioteki `libav`, gdyż wcześniej wykorzystywałem ją tylko do dekodowania audio w poprzednim projekcie z programowania II. w języku C++ oraz w projekcie komercyjnym jako skrypty wykorzystujące frontend `ffmpeg`.

Program można by było wzbogacić o opcje do konfiguracji formatu wyjściowego, co by sprawiło że mógłby być on uniwersalnym *klejem* do plików audio. Dodatkowo rozszerzenie go o strumienie video oraz napisów mogło by powiększyć możliwości klejenia. Takie zastosowanie wymagałoby zastosowania wsparcia akceleracji sprzętowej, która wymagała by testowania do kartach graficznych wielu producentów[8].

[7] lub flag kompilacyjnych związanych z `libmp3lame` w bibliotece `libav`, kwestie patentowe komplikują sprawę wsparcia tego kodowania.

[8] Firma Nvidia nie pozwala na transkodowanie więcej niż trzech strumieni na kartach konsumenckich, wymagana jest karta z rodziny co najmniej Quadro.