Universidade do Minho
Escola de Engenharia

Laboratórios de Informática III

Projeto

Primeira Fase

Grupo 25

Tiago Silva Figueiredo, A106856
Duarte Escairo Brandão Reis Silva, A106936
Luís António Peixoto Soares, A106932

1 Índice

2 Introdução

No âmbito da unidade curricular de Laboratórios de Informática III, foi-nos proposto realizar um projeto na linguagem de programação C para consolidarmos as aprendizagens relativas à linguagem, bem como, desenvolvermos novas aptidões, dado que nos foram introduzidas novas técnicas e ferramentas utilizadas na engenharia de *software*, tais como, encapsulamento e modularização, estruturas de dados mais complexas, *Valgrind*, bibliotecas como o *glib 2*.0 e *debugers*.

Para este projeto foi ainda utilizada a ferramenta GitHub para versionamento do código.

Nesta primeira fase foi trabalhada a estruturação do sistema de *streaming* de música proposto, sendo dividido em diversos campos como a leitura e filtragem de dados, validação desses mesmos dados e a resposta às 3 *queries* sugeridas.

3 Sistema

3.1 Estruturas de Dados

Neste projeto foi necessário recorrer à alocação de memória, e como tal, foi preciso decidir quais seriam as estruturas de dados mais adequadas para guardar toda a informação presente nos ficheiros CSV.

Os campos presentes em cada uma das entidades pré-estabelecidas (*artists*, *musics* e *users*) estão contidos em *structs* respetivas às mesmas, e foram todos descritos com variáveis do tipo *char*, char**, int, float* e *double*.

As primeiras entidades a serem lidas, filtradas e guardadas foram os *artists*, vindas do ficheiro *"artists.csv"*, e a estratégia utilizada para o seu armazenamento foram listas ligadas, definidas pelos elementos do grupo.

```
typedef struct artist_ll {
        Artist *artist;
        struct artist_ll *next
} LL_Artists;
```

Depois foi a vez das *musics*, vindas do ficheiro *"musics.csv"*, e a estrutura de dados utilizada para armazenar todas as músicas foi uma Hash Table.

Por fim, as informações dos *users*, que estavam no ficheiro *"users.csv"*, foram também guardadas numa Hash Table.

Para a implementação de ambas as Hash Tables que armazenam as *musics* e *users*, bem como as suas funções associadas, recorreu-se a biblioteca *glib 2.0*.

A escolha destas duas últimas estruturas deveu-se ao facto de nas *queries* 1 e 3 ser necessário fazer uma procura, como veremos mais à frente, e no caso das Hash Tables, o tempo de procura é constante, o que reduz o tempo de execução do programa.

3.2 Diagrama da Arquitetura do Sistema

Visando obedecer à arquitetura proposta, este projeto foi dividido em vários módulos, estando presentes módulos de entidades, estruturas de dados, gestão do sistema e módulos de *I/O*.

Nos módulos de entidades estão definidas todas as estruturas e funções referentes aos *artists, musics* e *users* (separadamente), incluindo *getters* e *setters*, que servem, acima de tudo para respeitar o encapsulamento.

Nos módulos de estruturas de dados, estão presentes as estruturas utilizadas para armazenar as informações de todas as entidades (separadamente), algumas delas já referidas em 3.1, bem como as funções que lidam diretamente com essas mesmas estruturas, como funções *add, search* e *free*.

Nos módulos de gestão de sistema são definidas funções que permitem gerir todas as estruturas de dados de todas as entidades.

Nos módulos de *I/O* está definida a função *parser* que trabalha diretamente com os ficheiros CSV, e está encarregue de colocar as informações das entidades nos seus devidos locais.

A seguir segue-se um diagrama que tem como objetivo ilustrar a arquitetura do projeto nesta primeira fase.

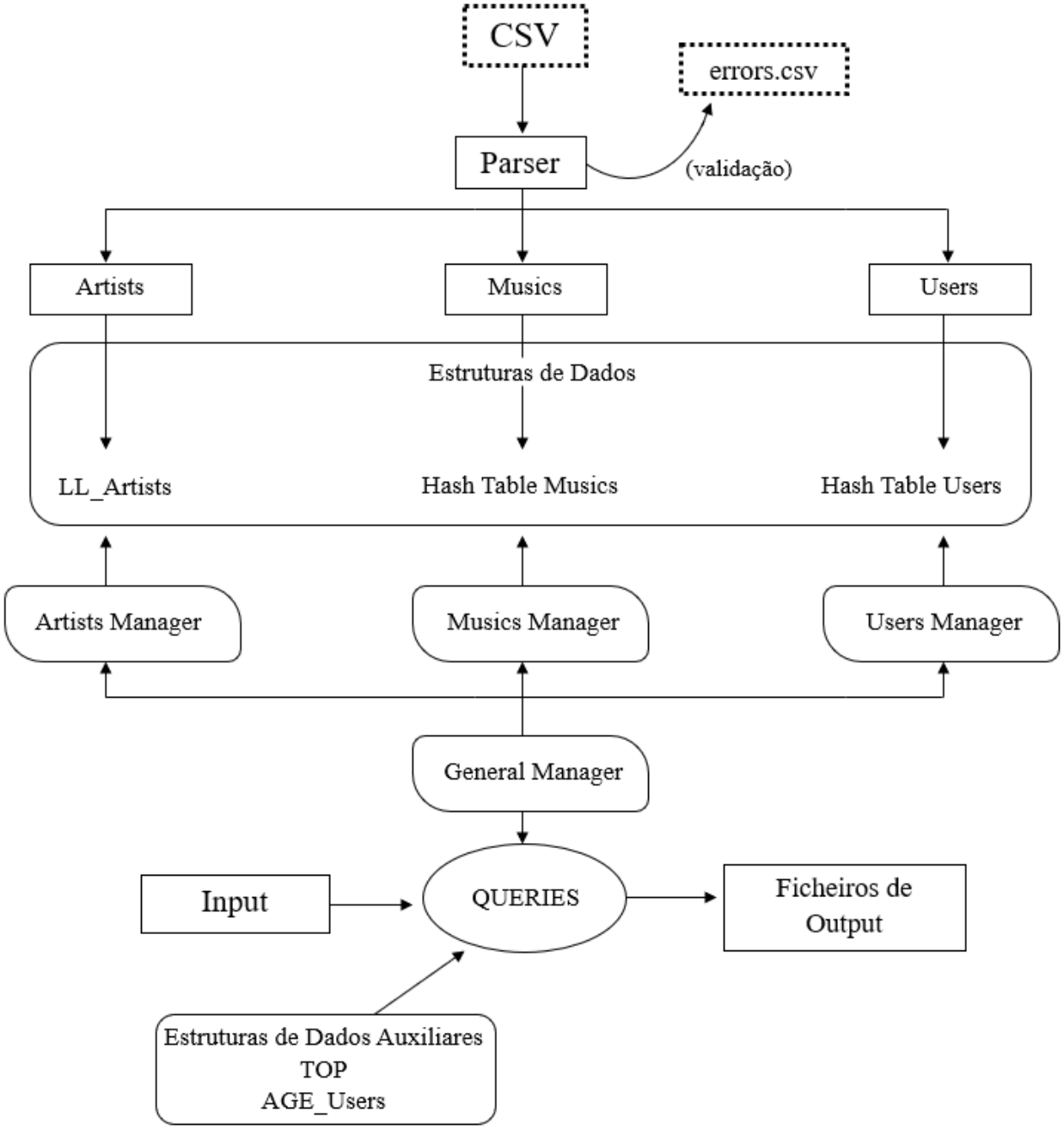


Figura 1: Diagrama da Arquitetura do Sistema

4 Discussão

4.1 Resolução das Queries

4.1.1 Querie 1

Na *querie* 1, é pedido para listar um resumo de um determinado utilizador, e colocar essas informações num ficheiro de *output*. Com o *id* que nos é passado como *input* temos de fazer uma procura dentro da estrutura de dados escolhida para armazenar entidades do tipo *user*.

Como decidimos utilizar uma Hash Table para armazenar os *users*, a resposta à questão torna-se rápida visto que o tempo de procura numa Hash Table é reduzido.

Após a procura estar concluída, temos então acesso às informações do *users* correspondente ao *id* passado, e basta apenas colocar as informações pedidas no ficheiro de *output*.

Uma das partes a ser listada é a idade do *user*, e dado que apenas temos posse da sua data de nascimento, usamos uma função auxiliar que calcula a sua idade a partir de uma *string*, no formato aaaa/mm/dd.

4.1.2 Querie 2

A *querie* 2 pede para colocar no ficheiro de *output* algumas informações dos top N artistas com maior discografia. O argumento obrigatório é o N e temos ainda um país como argumento opcional.

Esta *querie* foi mais trabalhosa de responder visto que, tivemos de recorrer a uma estrutura auxiliar

```
typedef struct top {
        char *artist;
        int duration;
        struct top *next
} TOP;
```

que contém os *ids* de todos os artistas, bem como o tempo total da sua discografia, em segundos.

Este TOP é ordenado por ordem decrescente, que faz com que apenas tenhamos de percorrer os primeiros N nodos da lista ligada. Para aceder às informações destes N artistas, é feita uma procura na lista ligada onde estão armazenados todos os artistas.

Quando no *input* o país está presente, a resolução é a mesma, apenas temos de verificar durante a procura na lista ligada dos artistas, se o país do artista a ser procurado no momento, coincide com o país do argumento.

Para facilitar a adição das discografias dos artistas é usada uma função auxiliar *duration_seg* que transforma uma *string* com o formato *hh/mm/ss* no número de segundos correspondente a esse tempo. No final para voltar a

converter esses segundos no formato original, utilizamos outra função *durarion_format*, que faz exatamente o inverso.

4.1.3 Querie 3

Na *querie* 3, o ficheiro de *output* terá de conter os géneros de músicas mais populares, ou seja, com mais *likes*, num determinado intervalo de idades passado como argumento.

Para responder a esta interrogação, começamos por percorrer a Hash Table que contêm os *users* e verificamos se a idade deles pertence ao intervalo de idades passado como argumento. Caso seja verdade, verifica-se o género de música, de cada uma das músicas pertencente às *liked_musics* do *user*, com auxílio da Hash Table onde são guardadas as músicas.

Depois de verificado o género da música, é adicionado +1 a um contador, que contém todos os géneros possíveis, juntamente com o respetivo número de *likes* associados, contador esse inicializado a 0.

Contudo, esta estratégia viu-se bastante ineficiente em termos de tempo de execução, então decidimos passar a usar uma estrutura auxiliar para responder a esta *querie*.

Essa estrutura

```
typedef age_User {
        int idade;
        int rock;
        int pop;
        int metal;
        int blues;
        int classical;
        int hip-hop;
        int country;
        int electronic;
        int reggae;
        int jazz;
        struct age_User *next;
} AGE_User;
```

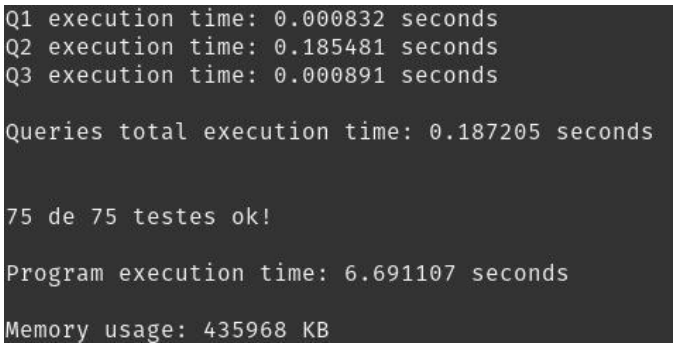
contém uma idade, o número total de *likes* de cada género, para aquela idade. Esta lista ligada é preenchida simultaneamente com a Hash Table que armazena os *users*.

Com esta estrutura auxiliar já pronta, bastou-nos apenas percorrê-la e ir acumulando os *likes* de cada género para as idades compreendidas no intervalo dado. Para o *output* estar de acordo com o esperado só tivemos de organizar estes valores por ordem decrescente.

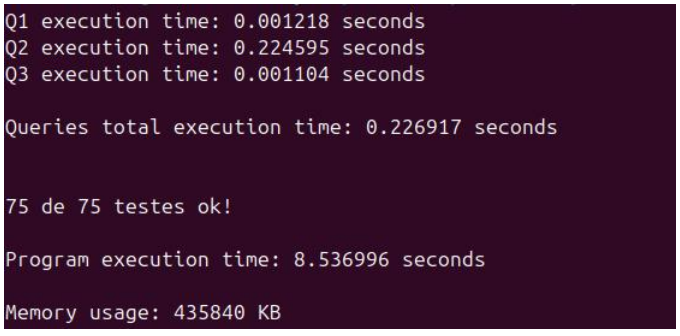
Com esta nova estratégia foram conseguidos tempos de execução muito inferiores aos anteriores, tendo passado de tempo de cerca de 50 segundo, para um tempo de 11,7 segundos.

4.2 Desempenho nas diferentes máquinas

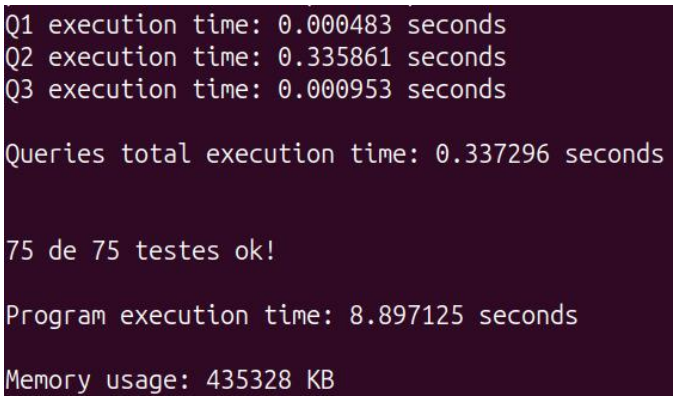
A seguir segue-se uns printscreens da execução do programa-teste desenvolvido no projeto, para mostrar os tempos de execução, memória usada e resultado das *queries* em cada uma das máquinas dos elementos do grupo.



Execução do programa-testes na máquina 1 (Tiago)



Execução do programa-testes na máquina 2 (Luís)



Execução do programa-testes na máquina 3 (Duarte)

Como se pode observar, os tempos de execução e a memória usada não são iguais para as três máquinas, isso deve-se ao facto de que cada uma delas ter componentes diferentes, o que influencia no desempenho da execução. Contudo, apesar de diferentes, os tempos e a memória não apresentam uma diferença muito significativa.

5 Conclusão

5.1 Dificuldades

Ao longo desta primeira fase do projeto deparamo-nos com algumas dificuldades, das quais três necessitaram de especial atenção, modularização e encapsulamento, *memory leaks* e memória utilizada.

5.1.1 Modularização e Encapsulamento

A estratégia inicial adotada pelo grupo, relativa à parte da modularização e encapsulamento, não estava completamente correta por dois grandes motivos; primeiro, o gestor geral do projeto ao invés de conter na sua estrutura os gestores das entidades (*artists, musics* e *users*), continha as estruturas de dados onde as entidades estavam armazenadas (listas ligadas e hash tables); segundo, em algumas funções do ficheiro *main*, eram passadas diretamente como argumentos, as estruturas de armazenamento ao invés de ser passado o gestor, o que quebrava o encapsulamento.

Ambos estes erros foram corrigidos após discutirmos com o docente do turno a nossa estratégia inicial.

5.1.2 Memory Leaks

Ao longo do projeto, sempre que verificávamos o nosso desempenho na plataforma de testes disponibilizada pelos docentes, a componente *memory leaks* esteve quase sempre com valores fora do esperado, chegando a estar com 85 MB.

Para ultrapassarmos esta dificuldade, foi preciso um trabalho minucioso de análise a vários ficheiros, juntamente com a ferramenta *Valgrind*.

Depois de várias tentativas, conseguimos obter os valores de *leaks* esperados.

5.1.3 Memória Utilizada

Outro dos problemas que tivemos dificuldades em manter dentro dos valores esperados na plataforma de testes foi o uso de memória, chegando a ter valores de 2200 MB, o que impedia o normal funcionamento da execução do programa na plataforma de testes.

Novamente, depois de uma revisão ao código, conseguimos baixar a memória usada para cerca de 436 MB.

5.2 Aprendizagens

Em conclusão, com o desenvolvimento desta primeira fase do projeto, ficou claro para nós uma significativa melhoria na compreensão da linguagem C, bem como dos métodos usados em engenharia de software.

Todas as dificuldades sentidas no decorrer do projeto foram essenciais para pudermos decidir quais os melhores caminhos a seguir para a resolução dos nossos problemas.

Estando esta primeira fase acabada, achamos que o início da segunda deva começar com uma possível reestruturação das funções *parser* e valida, visto que a sua eficiência pode e deve ser melhorada.