Trabalho banco de dados chave-valor

Integrantes: Nataniel Geraldo, Izabella de Castro, Helen Camila e Ícaro Emídio.

O objetivo do presente documento é a descrição básica do processo de desenvolvimento do banco de dados chave-valor, na disciplina de Sistemas Operacionais do curso de Engenharia de Software.

O programa foi desenvolvido conforme orientações do Prof. Pedro Penna, que forneceu os requisitos e diretrizes mínimas do trabalho. Na sequência será documentado as características gerais do trabalho:

1. **Linguagem de programação**
   a. O programa foi desenvolvido na linguagem C. Foi utilizada a IDE Visual Studio Code para escrita, compilação e teste do código.

2. **Sistema de versionamento Git**
   a. Foi utilizado o sistema de versionamento Git para controle de versões do projeto. O código fonte do programa foi hospedado no GitHub, e pode ser acessado pelo seguinte endereço:
      ➔ https://github.com/Nataniel93/simpledb

3. **Sistema de compilação do projeto**
   a. Para automatização da compilação do projeto, foi utilizado o Makefile. O arquivo contendo as diretivas necessárias no processo pode ser consultado no GitHub. Para compilação simplificada, basta executar o seguinte comando: **make**

4. **Implementação de teste**
   a. Os testes do programa foram implementados na “main”, e executados por threads.

Estrutura básica da solução

Conforme já explicitado, foi desenvolvido um banco de dados chave-valor, com a possibilidade de o usuário interagir com o programa por meio da linha de comandos. Neste sentido, seguem explicações dos parâmetros necessários:

➔ sort-key: representa a “tag” do tipo de dado a ser inserido, tal como:

Tag	Finalidade	Atributos
1	Cadastro de pessoas	Nome, CPF, Email
2	Cadastro de livros	Autor, Ano, Título, Editora
3	Cadastro de veículos	Marca, Ano, Modelo, Cor

Neste projeto, foi desenvolvido a lógica apenas para “Cadastro de pessoas”.

➔ value: representa o dado a ser efetivamente cadastrado no banco de dados. Neste caso, foi utilizada a estrutura JSON que permite a inclusão de diversos atributos. Segue modelo de JSON utilizado, com os atributos de cadastro de pessoas:

{'nome':'joao','cpf':'123456789','email':'joao@joao.com.br'}

➔ key: representa a chave de identificação dos dados inseridos no banco de dados.

No caso do cadastro de pessoas, a chave é o CPF.

Após definição dos itens acima, é possível citar as operações suportadas pela interface de linha de comandos:

--insert=<sort-key,value>
--search=<key>
--remove=<key>
--update=<key,sort-key,value>

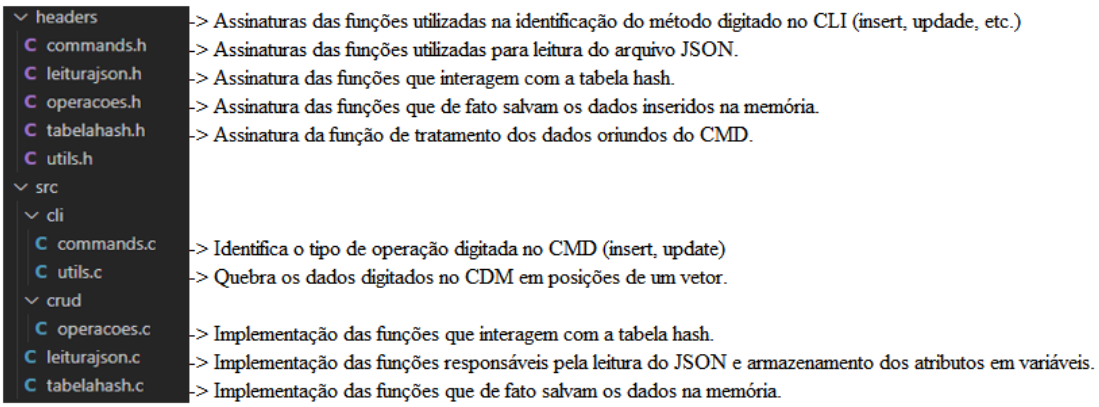
Para alocação dos dados na memória, foi utilizada a estrutura de tabela hash com lista encadeada. O CPF foi utilizado na função de espalhamento, tendo em vista a alocação dos dados na posição específica da tabela. As colisões foram tratadas justamente pela utilização da lista encadeada. Sendo assim, caso uma determinada inserção tente ser alocado na mesma posição da tabela que já possui dados, será encadeada em um novo nó da lista. As estruturas utilizadas na tabela hash foram:

```
typedef struct no
{
Pessoa pessoa;
struct no *proximo;
} No;
```

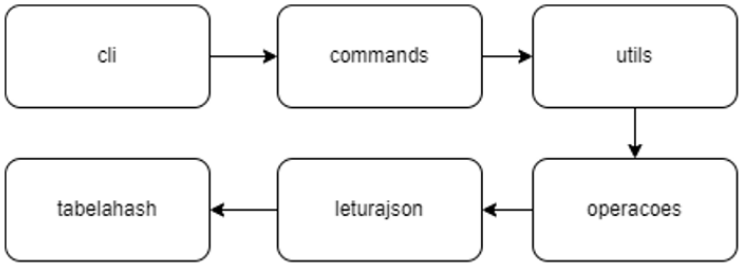
```
typedef struct
{
No *inicio;
int tam;
} Lista;
```

```
typedef struct
{
long unsigned cpf;
char nome[50];
char email[50];
} Pessoa;
```

Ademais, o código foi modularizado e arquivos específicos foram geradas:



Com o intuito de esclarecer o “caminhamento” da operação de inserção, por exemplo, foi elaborado o esquema a seguir:



O projeto inicialmente desenvolvido suportava de fato a interação com linha de comandos, sendo realizados inclusive testes. Entretanto, devido as limitações de tempo, não foi possível criar uma interface separada para capturar os dados inseridos pelo usuário. Entretanto, foi disponibilizado no projeto um arquivo chamado “testeInicial” que demonstra como seriam tratados os dados em caso de leitura da CLI, por meio da “argv”.

Segue abaixo print da tela após execução do arquivo citado:

```
nataniel@LAPTOP-UCAPTFOF:~/TrabalhoSO/simpledb$ ./testeInicial --insert="1,{'nome':'Joao','cpf':'111111111','email':'joac@joao.com.br'}"

DATA[0] -> 1

DATA[1] -> {'nome':'Joao','cpf':'111111111','email':'joac@joao.com.br'}

--------------------TABELA-----------------------
967 Lista tamanho: 1
       Nome: Joac@    CPF: 111111111  Email: joac@joao.com.br@R@
--------------------FIM TABELA----------------------

Pessoa inserida com sucesso!
nataniel@LAPTOP-UCAPTFOF:~/TrabalhoSO/simpledb$ ./testeInicial --update=""

Eh Update

nataniel@LAPTOP-UCAPTFOF:~/TrabalhoSO/simpledb$ ./testeInicial --search=""

Eh Search

nataniel@LAPTOP-UCAPTFOF:~/TrabalhoSO/simpledb$ ./testeInicial --remove=""

Eh Remove

nataniel@LAPTOP-UCAPTFOF:~/TrabalhoSO/simpledb$
```

Diante disto, o projeto final conta apenas com os testes abaixo descritos, que demonstram de certa forma a robustez da solução.

Testes

Os testes do programa foram realizados no arquivo “main”. Foram realizadas operações de inserir, atualizar, remover e pesquisar, tal como segue:

➔ Inserir

1. "{'nome':'Joao','cpf':'111111111','email':'joao@joao.com.br'}"
2. "{'nome':'Ricardo','cpf':'222222222','email':'ricardo@ricardo.com.br'}"
3. "{'nome':'Maria','cpf':'333333333','email':'maria@maria.com.br'}"
4. "{'nome':'Jose','cpf':'444444444','email':'jose@jose.com.br'}"
5. "{'nome':'Jaqueline','cpf':'555555555','email':'jaqueline@jaqueline.com.br'}"

➔ Atualizar

1. "{'nome':'Icaro','cpf':'111111111', 'email':'icaro@icaro.com.br'}"
2. "{'nome':'Nataniel','cpf':'222222222', 'email':'nataniel@nataniel.com.br'}"
3. "{'nome':'Helen','cpf':'444444444', 'email':'helen@helen.com.br'}"

➔ Remover

1. remover_pessoa(333333333)
2. remover_pessoa(555555555)

➔ Pesquisar

1. buscar_pessoa(333333333)

No final da execução do teste, a tabela foi impressa com as três pessoas que foram atualizadas e não foram excluídas. Sendo assim, o teste foi executado com sucesso, tal como segue:

```
nataniel@LAPTOP-UCAPTFOF:~/TrabalhoSO/simpledb$ ./simpledb
initializing...
thread 1 running...
thread 2 running...
Operação de remocao efetuado com sucesso!
Operação de remocao efetuado com sucesso!
thread 2 exiting...
Operação de insercao efetuado com sucesso!
Operação de insercao efetuado com sucesso!
Operação de insercao efetuado com sucesso!
Operação de insercao efetuado com sucesso!
Operação de insercao efetuado com sucesso!
Operação de atualizacao efetuado com sucesso!
Operação de atualizacao efetuado com sucesso!
Operação de atualizacao efetuado com sucesso!
        Nome: Maria     CPF: 333333333  Email: maria@maria.com.br
Operação de pesquisa efetuado com sucesso!
thread 1 exiting...

---------------------TABELA-------------------------
796 Lista tamanho: 1
        Nome: Helen     CPF: 444444444  Email: helen@helen.com.br
910 Lista tamanho: 1
        Nome: Nataniel  CPF: 222222222  Email: nataniel@nataniel.com.br
967 Lista tamanho: 1
        Nome: Icaro     CPF: 111111111  Email: icaro@icaro.com.br
---------------------FIM TABELA----------------------
shutting down...
nataniel@LAPTOP-UCAPTFOF:~/TrabalhoSO/simpledb$
```

Conclusão

Por fim, concluímos que embora não tenha sido possível a implementação de todos os requisitos originais do trabalho, acreditamos que a solução procurou atender as convenções de boas práticas de programação, tais como modularização e documentação.