Ciências ULisboa | Informática

2015/16

Sistemas Distribuídos

Projeto 1

1. Descrição geral

A componente teórico-prática da disciplina de sistemas distribuídos está divida em cinco projetos, sendo que a realização de cada um deles é necessária para a realização do projeto seguinte. Por essa razão, **é muito importante que consigam ir cumprindo os objetivos de cada projeto, de forma a não hipotecar os projetos seguintes**.

O objetivo geral do projeto será concretizar um serviço de armazenamento de pares chave-valor similar ao utilizado pela *Amazon* para dar suporte aos seus serviços Web [1]. A estrutura de dados utilizada para armazenar esta informação é uma **tabela *hash*** [2], dada a sua elevada eficiência ao nível da pesquisa. Uma função *hash* é usada para transformar cada chave num índice (*slot*) de um array (*bucket*) onde ficará armazenado o par chave-valor. Idealmente, todas as chaves seriam mapeadas para um *slot* específico, mas tal nem sempre é possível e podem ocorrer *colisões*, quando chaves diferentes são mapeadas no mesmo *slot*. Para lidar com as colisões vai utilizar-se a técnica de *chaining*, ilustrada na Figura 1.

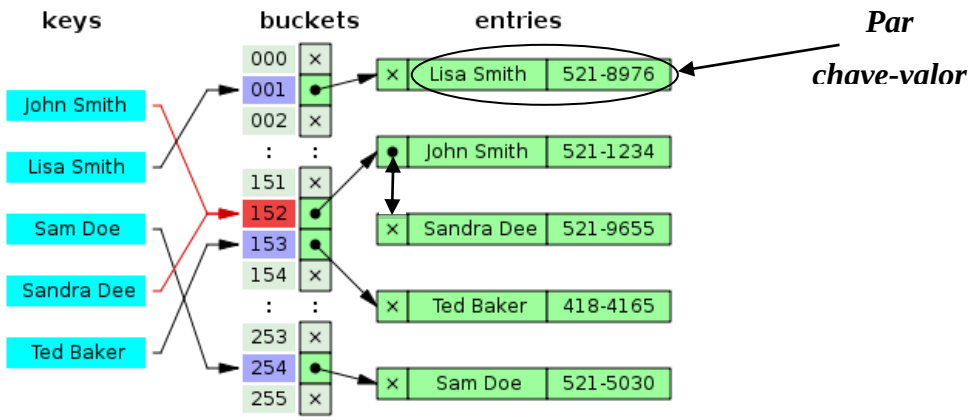

Figura 1. Tabela *hash* com *chaining* [2]

Nesta técnica as colisões são resolvidas fazendo com que cada *slot* do *bucket* seja um ponteiro para uma lista ligada. Esta contém todos os pares chave-valor cujo *hash* resultou no mesmo índice. No exemplo da figura, podemos verificar que a função *hash* executada sobre as chaves "John Smith" e "Sandra Dee" resultou no índice 152. Por essa razão, a lista ligada apontada pelo índice 152 contém esses dois pares.

2. Descrição específica

O projeto 1 consiste na concretização em C [3] de alguns módulos fundamentais:

(i) Definição do tipo de dados a armazenar;
(ii) Definição de uma entrada da tabela *hash*;
(iii) Criação de uma lista ligada.

Para cada um destes módulos, é fornecido um ficheiro *.h* com os cabeçalhos das funções, que **não pode ser alterado**. As concretizações das funções definidas nos ficheiros *X.h* devem ser feitas num ficheiro *X.c*, utilizando os algoritmos e métodos que o grupo achar convenientes.

Se o grupo entender necessário, ou se for pedido, também pode criar um ficheiro *X-private.h* para acrescentar outras definições, a incluir no ficheiro *X.c*. Os ficheiros *.h* apresentados neste documento bem como alguns testes para as concretizações realizadas, serão disponibilizados na página da disciplina.

2.1. Definição do elemento de dados

A primeira tarefa consiste em definir o formato dos dados que serão armazenados na tabela *hash*, no servidor. Para isso, é dado o ficheiro *data.h* que define a estrutura que contém os dados e respetiva dimensão, bem como funções para a sua criação e destruição.

```
#ifndef _DATA_H
#define _DATA_H

/* Estrutura que define os dados.
 */
struct data_t {
      int datasize; /* Tamanho do bloco de dados */
      void *data;   /* Conteúdo arbitrário */
};

/* Função que cria um novo elemento de dados data_t e reserva a memória
 * necessária, especificada pelo parâmetro size
 */
struct data_t *data_create(int size);

/* Função idêntica à anterior, mas que inicializa os dados de acordo com
 * o parâmetro data.
 */
struct data_t *data_create2(int size, void * data);

/* Função que destrói um bloco de dados e liberta toda a memória.
 */
void data_destroy(struct data_t *data);

/* Função que duplica uma estrutura data_t.
 */
struct data_t *data_dup(struct data_t *data);

#endif
```

2.2. Definição de uma entrada

Definidos que estão os dados (o *valor*), é agora necessário criar uma entrada para a tabela *hash*, definida como um par {chave, valor}.

Para este efeito, é dado o ficheiro *entry.h* que define a estrutura de uma entrada na tabela, bem como funções para a sua criação e destruição. Estas funções devem, naturalmente, utilizar as que estão implementadas no módulo `data`, onde, e se necessário.

```
#ifndef _ENTRY_H
#define _ENTRY_H

#include "data.h"

/* Esta estrutura define o par {chave, valor} para a tabela
 */
struct entry_t {
      char *key;  /* string, (char* terminado por '\0') */
      struct data_t *value; /* Bloco de dados */
};
```

```
/* Função que cria um novo par {chave, valor} (isto é, que inicializa
 * a estrutura e aloca a memória necessária).
 */
struct entry_t *entry_create(struct data_t *data);

/* Função que destrói um par {chave-valor} e liberta toda a memória.
 */
void entry_destroy(struct entry_t *entry);

/* Função que duplica um par {chave, valor}.
 */
struct entry_t *entry_dup(struct entry_t *entry);

#endif
```

2.3. Lista ligada

A última tarefa do projeto 1 consiste em implementar um módulo de criação e destruição de **listas ligadas ordenadas** (as quais vão "armazenar" os pares chave-valor). A ordenação da lista deve ser **por ordem crescente** das chaves alfanuméricas contidas nas entradas do tipo `enty_t`. O ficheiro *list.h* define as estruturas e as funções a serem concretizadas para atingir esse objetivo.

```
#ifndef _LIST_H
#define _LIST_H

#include "entry.h"

struct list_t; /*A definir pelo grupo em list-private.h*/

/* Cria uma nova lista. Em caso de erro, retorna NULL.
 */
struct list_t *list_create();

/* Elimina uma lista, libertando *toda* a memoria utilizada pela
 * lista.
 */
void list_destroy(struct list_t *list);

/* Adiciona uma entry na lista. Como a lista deve ser ordenada,
 * a nova entry deve ser colocada no local correto.
 * Retorna 0 (OK) ou -1 (erro)
 */
int list_add(struct list_t *list, struct entry_t *entry);

/* Elimina da lista um elemento com a chave key.
 * Retorna 0 (OK) ou -1 (erro)
 */
int list_remove(struct list_t *list, char* key);

/* Obtem um elemento da lista que corresponda à chave key.
 * Retorna a referência do elemento na lista (ou seja, uma alteração
 * implica alterar o elemento na lista).
 */
struct entry_t *list_get(struct list_t *list, char *key);

/* Retorna o tamanho (numero de elementos) da lista
 * Retorna -1 em caso de erro.  */
int list_size(struct list_t *list);
```

```
/* Devolve um array de char * com a cópia de todas as keys da
 * tabela, e um último elemento a NULL.
 */
char **list_get_keys(struct list_t *list);

/* Liberta a memoria reservada por list_get_keys.
 */
void list_free_keys(char **keys);

#endif
```

Caso seja necessário, no ficheiro `list-private.h` deverão ser definidas funções adicionais necessárias à implementação em `list.h`.

3. Entrega

A entrega do projeto 1 consiste em colocar todos os ficheiros do projeto, bem como o ficheiro README mencionado abaixo, num ficheiro com compressão no formato ZIP. Este ficheiro será depois entregue na página da disciplina, no moodle da FCUL.

O ficheiro ZIP deverá conter uma diretoria cujo nome é **`grupoXX`**, onde **`XX`** é o número do grupo. Nesta diretoria serão colocados:

- o ficheiro README, onde os alunos devem explicar como executar o projeto e incluir outras informações que julguem necessárias (e.g., limitações na implementação);
- diretorias adicionais para armazenar os ficheiros *.c* e *.h* correspondentes a cada módulo;
- um ficheiro `Makefile` que permita a correta compilação de todos os ficheiros entregues. **Se não for incluído um `Makefile`, se o mesmo não compilar os ficheiros fonte, ou se houver erros de compilação (isto é, se não forem criados os ficheiros objeto), o trabalho é considerado nulo**.
  Na página da cadeira podem encontrar vídeos e documentos do utilitário `make` e dos ficheiros `Makefile` (cortesia da disciplina de Sistemas Operativos).

Todos os ficheiros entregues devem começar com três linhas de comentários a dizer o número do grupo e o nome e número de seus elementos.

O prazo de entrega é domingo, dia 4/10/2015, até às 22:00hs.

4. Bibliografia

[1] Giuseppe DeCandia et al. *Dynamo: Amazon's Highly Available Key-value Store*. Proc. of the 21st Symposium on Operating System Principles – SOSP'07. pp. 205-220. Out. de 2007.

[2] Wikipedia. *Hash Table*. http://en.wikipedia.org/wiki/Hash_table.

[3] B. W. Kernighan, D. M. Ritchie, *C Programming Language*, 2nd Ed, Prentice-Hall, 1988.