Ciências ULisboa | Informática

2015/16

Sistemas Distribuídos

Projeto 2

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* (ver enunciado do projeto 1).

No projeto 1 foram definidas algumas estruturas de dados e implementadas várias funções para lidar com a manipulação dos dados que vão ser armazenados na tabela, bem como para implementar a técnica de *chaining* utilizando uma lista ligada. No projeto 2 o objetivo é implementar as funções necessárias para construir a tabela *hash* e para codificar e descodificar estruturas complexas em mensagens. Esta última tarefa vai servir para possibilitar a comunicação entre cliente e servidor nos projetos seguintes.

2. Descrição específica

O projeto 2 consiste na concretização em C de dois módulos fundamentais:

(i) Criação de uma estrutura de dados (tabela *hash*) onde serão armazenados os pares chave-valor nos servidores

(ii) A codificação e descodificação de estruturas complexas em mensagens, para possibilitar a comunicação entre cliente e servidor no projeto seguinte.

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. Tabela hash

A tabela hash deve armazenar os dados e oferecer operações do tipo **put**, **get**, **del**, **update** e **size**. O ficheiro table.h define as estruturas e as funções a serem concretizadas neste módulo.

Como referido atrás, para resolver o problema de colisões, deve utilizar-se na concretização da tabela hash a lista ligada criada no projeto anterior. A ideia é ter um *array* (um bucket no enunciado anterior) de tamanho fixo *n* (definido pelo parâmetro da função *table_create*), com cada slot apontando para uma lista. Quando é necessário executar uma inserção ou uma busca,

primeiro aplica-se uma função hash para descobrir em que lista (slot) a entrada será inserida/procurada, e depois executa-se a função desejada na lista em questão.

Deve ser usada a seguinte função hash para mapear a chave em índices de listas (inteiros entre um 0 e *n*-1):

- Para chaves com tamanho até 6 caracteres (inclusive), soma-se o valor ASCII de todos os caracteres da chave e depois calcula-se o resto da divisão da soma por *n*.
- Para chaves maiores (mais de 6 caracteres), soma-se o valor ASCII dos primeiros 3 caracteres da chave e dos 3 últimos, e calcula-se o resto da divisão da soma por *n*.

O ficheiro *table.h* que define as estruturas e as funções a serem concretizadas neste módulo é o seguinte:

```
#ifndef _TABLE_H
#define _TABLE_H

#include "list-private.h"


struct table_t; /* A definir pelo grupo em table-private.h */

/* Função para criar/inicializar uma nova tabela hash, com n
 * linhas(n = módulo da função hash)
 */
struct table_t *table_create(int n);

/* Libertar toda a memória ocupada por uma tabela.
 */
void table_destroy(struct table_t *table);

/* Função para adicionar um par chave-valor na tabela.
 * Os dados de entrada desta função deverão ser copiados.
 * Devolve 0 (ok) ou -1 (out of memory, outros erros)
 */
int table_put(struct table_t *table, char *key, struct data_t *value);

/* Função para substituir na tabela, o valor associado à chave key.
 * Os dados de entrada desta função deverão ser copiados.
 * Devolve 0 (OK) ou -1 (out of memory, outros erros)
 */
int table_update(struct table_t *table, char *key, struct data_t *value);

/* Função para obter da tabela o valor associado à chave key.
 * A função deve devolver uma cópia dos dados que terão de
 * ser libertados no contexto da função que chamou table_get.
 * Devolve NULL em caso de erro.
 */
struct data_t *table_get(struct table_t *table, char *key);

/* Função para remover um par chave valor da tabela, especificado
 * pela chave key, libertando a memória associada a esse par.
 * Devolve: 0 (OK), -1 (nenhum tuplo encontrado; outros erros)
 */
int table_del(struct table_t *table, char *key);

/* Devolve o número de elementos na tabela.
 */
int table_size(struct table_t *table);

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

```
/* Liberta a memória alocada por table_get_keys().
 */
void table_free_keys(char **keys);

#endif
```

2.2. *Marshaling* e *unmarshaling* de mensagens

Este módulo do projeto consiste em transformar uma estrutura de dados complexa num formato que possa ser enviado pela rede (isto é, convertê-la num formato "unidimensional"), e vice-versa. O ficheiro *message.h* define as estruturas e as funções a serem concretizadas neste módulo.

Note-se que neste módulo vão ser usadas uniões [3, 4]. No caso da *struct message_t* abaixo, a ideia é que, dependendo do código da operação representada na mensagem (campo *opcode*), uma variável de tipo diferente possa ser utilizada.

```
#ifndef _MESSAGE_H
#define _MESSAGE_H

#include "data.h"
#include "entry.h"

/* Define os possíveis opcodes da mensagem */
#define OC_SIZE         10
#define OC_DEL          20
#define OC_UPDATE       30
#define OC_GET          40
#define OC_PUT          50

/* Define códigos para os possíveis conteúdos da mensagem */
#define CT_RESULT       10
#define CT_VALUE        20
#define CT_KEY          30
#define CT_KEYS         40
#define CT_ENTRY        50

/* Estrutura que representa uma mensagem genérica a ser transmitida.
 * Esta mensagem pode ter vários tipos de conteúdos.
 */
struct message_t {
      short opcode; /* código da operação na mensagem */
      short c_type; /* tipo do conteúdo da mensagem */
      union content_u {
            int result;
            struct data_t *data;
            char *key;
            char **keys;
            struct entry_t *entry;
      } content;    /* conteúdo da mensagem */
};

/* Converte o conteúdo de uma message_t num char *, retornando o tamanho do
 * buffer alocado para a mensagem serializada como um array de bytes, ou -1
 * em caso de erro.
 * A mensagem serializada numa sequência de bytes, deve ter o seguinte
 * formato:
 *
 * OPCODE         C_TYPE
 * [2 bytes]      [2 bytes]
 *
 * a partir daí, o formato difere para cada tipo de conteúdo (c_type):
```

```
 * CT_ENTRY       KEYSIZE(KS)       KEY         DATASIZE(DS)      DATA
 *                [2 bytes]         [KS bytes]  [4 bytes]         [DS bytes]
 * CT_KEY         KEYSIZE(KS)       KEY
 *                [2 bytes]         [KS bytes]
 * CT_KEYS        NKEYS             KEYSIZE(KS) KEY         ...
 *                [4 bytes]         [2 bytes]   [KS bytes]  ...
 * CT_VALUE       DATASIZE(DS)      DATA
 *                [4 bytes]         [DS bytes]
 * CT_RESULT      RESULT
 *                [4 bytes]
 *
 * Notar que o `\0´ no fim da string e o NULL no fim do array de
 * chaves não são enviados nas mensagens.
 */
int message_to_buffer(struct message_t *msg, char **msg_buf);

/* Transforma uma mensagem no array de bytes, buffer, para
 * uma struct message_t*
 */
struct message_t *buffer_to_message(char *msg_buf, int msg_size);

/* Liberta a memoria alocada na função buffer_to_message
 */
void free_message(struct message_t *msg);

#endif
```

3. Entrega

A entrega do projeto 2 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 18/10/2015, 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.

[4] http://markburgess.org/CTutorial/CTutorial.html#Unions