Departamento de Informática da
Faculdade de Ciências da
Universidade de Lisboa

Sistemas Distribuídos

2012/13

Projeto 1

Versão 1.1

1. Descrição geral

A parte teórico-prática da cadeira de sistemas distribuídos está divida em cinco projetos, sendo que cada um deles corresponde a partes necessárias 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.

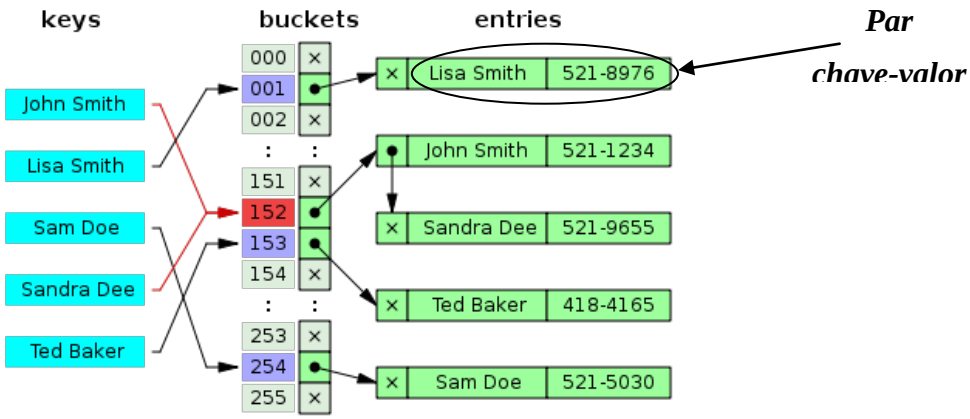


Figura 1. Tabela *hash* com *chaining* [2]

Nesta técnica as colisões são resolvidas fazendo com que cada *slot* do *bucket* seja um ponteiro para uma lista encadeada. 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 encadeada apontada pelo índice 152 contém esses dois pares.

2. Descrição específica

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

(i) Definição do bloco de dados que será depois associado a uma chave (o *valor*)
(ii) Definição e criação de uma entrada *(chave, valor)*
(iii) Criação de uma lista encadeada que armazena entradas com chaves e valores

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 entender necessário, o grupo pode também criar um ficheiro *X-private.h* para acrescentar outras definições, a incluir no ficheiro *X.c*. Os ficheiros *.h* apresentados neste documento e alguns testes para as concretizações realizadas serão disponibilizados na página da cadeira.

2.1. Blocos de dados

A primeira tarefa é definir o formato dos dados que serão armazenados no servidor (e que ficarão associados a uma chave). Para isso, é dado o ficheiro *data.h* que define a estrutura que contém o bloco de dados e o seu tamanho, bem como funções para a sua criação e destruição.

```
#ifndef _DATA_H
#define _DATA_H

/* Estrutura que contem os dados, isto e, o valor do par
 * (chave, valor), juntamente com o seu tamanho.
 */
struct data_t {
     int datasize;   /* Tamanho do bloco de dados data */
     void *data; /* Conteúdo arbitrario */
};

/* Funcao que cria um novo bloco de dados (isto e, que inicializa
 * a estrutura e aloca a memoria necessaria).
 */
struct data_t *data_create(int size);

/* Funcao identica a anterior, mas com uma assinatura diferente.
 */
struct data_t *data_create2(int size, void *data);

/* Funcao que destroi um bloco de dados e liberta toda a memoria.
 */
void data_destroy(struct data_t *data);

/* Funcao que duplica um bloco de dados. Quando se criam duplicados
 * e necessario efetuar uma COPIA dos dados (e nao somente alocar a
 * memoria necessaria).
 */
struct data_t *data_dup(struct data_t *data);

#endif
```

Obs: *void** é um tipo que representa um apontador para um bloco genérico de dados. Na prática é equivalente a *char**, mas é usado aqui para evitar confusão com *strings*.

2.2. Par (chave, valor)

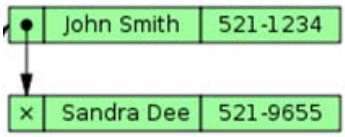
Definidos que estão os dados (o *valor*), agora é necessário criar uma entrada para a tabela, isto é, um par (chave, valor). Para isso, é dado o ficheiro *entry.h* que define a estrutura que contém o par (chave, valor), bem como funções para a sua criação e destruição. Estas funções devem, naturalmente, utilizar as funções 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') */
     long long timestamp; /* Por agora 0, a ser usado no proj. 5 */
     struct data_t *value; /* Bloco de dados */
};

/* Funcao que cria um novo par chave-valor (isto e, que inicializa
 * a estrutura e aloca a memoria necessaria).
 */
struct entry_t *entry_create(char *key, struct data_t *data);

/* Funcao que destroi um par chave-valor e liberta toda a memoria.
 */
void entry_destroy(struct entry_t *entry);

/* Funcao que duplica um par chave-valor. */
struct entry_t *entry_dup(struct entry_t *entry);


#endif
```

2.3. Lista encadeada

A última tarefa do projeto 1 consiste em implementar um módulo de criação e destruição de **listas encadeadas ordenadas** (as quais vão "armazenar" os pares chave-valor). A ordenação da lista deve ser por ordem crescente das chaves alfanuméricas dos pares chave-valor inseridos. 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.
 * Retorna 0 (OK) ou -1 (erro)
 */
int 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 com a chave key.
 * Retorna a referencia do elemento na lista (ou seja, uma alteração
 * implica alterar o elemento na lista).
 * Nota: a função list_remove ou list_destroy deve ser
 * a responsável por libertar a memoria ocupada pelo elemento.
 */
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
 * lista, com um ultimo elemento a NULL.
 */
char **list_get_keys(struct list_t *list);

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

#endif
```

3. Entrega

A entrega do projeto 1 consiste em colocar todos os ficheiros .c e .h do projeto na diretoria **$HOME/areas_de_grupo/sdNNN/projeto1** (note-se que não é *projecto-1*, nem *proj1*, nem *projecto1*). Se acharem necessário, podem incluir um documento de suporte, com um máximo de duas páginas, a explicar resumidamente a vossa implementação ou alguns detalhes que julguem úteis.

Os alunos devem também incluir 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).

O prazo de entrega é domingo, dia 7/10/2011, até às 22:00hs.

Posteriormente, cada grupo ficará sem acesso de escrita à diretoria de entrega.

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://www.iu.hio.no/~mark/CTutorial/CTutorial.html#Unions