Ciências ULisboa | Informática

2015/16

Sistemas Distribuídos

Projeto 3

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 hipotecarem 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 implementaram-se as funções necessárias para construir a tabela *hash* e para codificar e descodificar estruturas complexas em mensagens.

No projeto 3 vamos concretizar a tabela *hash* num servidor, oferecendo ao cliente uma interface similar à que foi concretizada no projeto 2 para a tabela. Isto implica que:

1. o cliente irá invocar operações, que serão transformadas em mensagens e enviadas pela rede até ao servidor;
2. este, por sua vez interpretará essas invocações, e;
    a. realizará as operações correspondentes na sua tabela local;
    b. enviará posteriormente a resposta transformada em mensagem, ao cliente;
3. por sua vez, o cliente interpretará a mensagem de resposta, e;
4. procederá de acordo com o resultado, ficando de seguida pronto para a próxima operação.

A concretização do projeto 3 será efetuada combinando o código desenvolvido nos projetos anteriores com as técnicas para comunicação por *sockets* TCP. Espera-se uma grande fiabilidade por parte do servidor, portanto não pode haver condições de erro não verificadas ou gestão de memória ineficiente a fim de evitar que este sofra um *crash*, o que deixaria todos os clientes sem a tabela partilhada (no caso da Amazon, se o serviço que mantém as cestas de compras dos clientes não funciona, a empresa não vende, perde milhões de dólares por hora… e os programadores são despedidos!).

2. Programas a implementar

Devem implementar-se dois programas, a serem executados da seguinte forma:

- **table-server** <port> <*n_lists*>

  <*port*> é o número do porto ao qual o servidor se deve ligar (fazer *bind*).

  <*n_lists*> é o número de listas usado na criação da tabela no servidor.

- **table-client** <*server*>:<*port*>

  <*server*> é o endereço ou nome do servidor da tabela.

  <*port*> é o número do porto onde o servidor está à espera de ligações.

3. Cliente

O cliente usa as funcionalidades definidas nos projeto 1 e 2, juntamente com duas partes adicionais:

- Aplicação cliente interativa (table_client.c) com a função main()
- Biblioteca de comunicação (network_client.h/c)

3.1. table_client.c

A aplicação cliente a realizar é um programa interativo simples que aceita um comando (uma linha) do utilizador no *stdin*, invoca a função necessária da biblioteca de comunicação (secção 3.2), imprime a resposta no ecrã, e volta a aceitar o próximo comando.

Depois de receber os dados do utilizador, este módulo tem então de preencher uma estrutura *message_t* com o *opcode* (tipo da operação) e os campos usados de acordo com o conteúdo requerido pela mensagem e vai passar esta estrutura para o módulo de comunicação, recebendo depois uma resposta deste módulo (ver próxima secção). Cada comando vai ser inserido pelo utilizador numa única linha, havendo as seguintes alternativas:

- put <key> <data>
- get <key>
- update <key> <data>
- del <key>
- size
- quit

Note que a aplicação cliente vai usar o campo 'data' da estrutura `data_t` do projeto 1 para guardar a *string* <data> introduzida pelo utilizador nos comandos put e update (que pode conter espaços, i.e., é a *string* completa após a chave <key>). No entanto, todos os restantes módulos (*network_client.c* e servidor) devem suportar dados arbitrários neste campo (e.g., imagens, som, etc.).

Dica: uma boa forma de ler e tratar os comandos inseridos é usando as funções *fgets* e *strtok*.

3.2. Biblioteca de comunicação do cliente: network_client.c

O módulo *network-client.c* vai serializar a mensagem, utilizando a função *message_to_buffer* do projeto 2, enviá-la ao servidor, e esperar pela resposta. A biblioteca de comunicação *network_client.c* tem a seguinte interface:

```
#ifndef _NETWORK_CLIENT_H
#define _NETWORK_CLIENT_H

#include "message-private.h"

struct server_t; //a definir pelo grupo em network_client-private.h

/* Esta função deve:
 *  - estabelecer a ligação com o servidor;
 *  - address_port é uma string no formato <hostname>:<port>
 *    (exemplo: 10.10.10.10:10000)
 *  - retornar toda a informacão necessária (e.g., descritor da
 *    socket) na estrutura server_t
 */
struct server_t *network_connect(const char *address_port);

/* Esta função deve
 * - Obter o descritor da ligação (socket) da estrutura server_t;
 * - enviar a mensagem msg ao servidor;
 * - receber uma resposta do servidor;
 * - retornar a mensagem obtida como resposta ou NULL em caso
 *   de erro.
 */
struct message_t *network_send_receive(struct server_t *server,
                                       struct message_t *msg);

/* A função network_close() deve fechar a ligação estabelecida por
* network_connect(). Se network_connect() alocou memória, a função
* deve libertar essa memória.
*/
int network_close(struct server_t *server);

#endif
```

As mensagens na rede devem usar o formato definido no projeto 2. Para facilitar a receção, o cliente deve executar os três passos seguintes:

a) usar *message_to_buffer* para transformar a mensagem num *buffer* e determinar o número de bytes da mensagem serializada;
b) enviar primeiro um inteiro (4 bytes) em formato de rede indicando ao servidor o tamanho da mensagem serializada que enviará de seguida, e;
c) enviar o *buffer* (que será interpretado pelo servidor através de *buffer_to_message*).

Na resposta, o servidor executa os mesmos passos, ou seja, envia primeiro o tamanho num inteiro (4 bytes), e só depois o *buffer* com a mensagem de resposta serializada.

4. Servidor

O servidor a implementar usa todas as funcionalidades dos projetos 1 e 2, além da aplicação servidor (*table_server.c*) com a função *main()*.

4.1. table_server.c

O servidor concretiza uma tabela que pode ser acedida no porto definido na linha de comando, e deve aceitar ligações neste porto e em qualquer interface. Os servidores deverão

suportar apenas um cliente de cada vez. Como o servidor apenas necessita de tratar um pedido de cada vez, não será necessário recorrer à subdivisão do programa em *threads* ou processos filho (e.g., através da chamada de sistema *fork()*).

5. Observações

Algumas observações e dicas úteis:

- Recomenda-se a criação de funções *read_all* e *write_all*, que vão receber e enviar *arrays* de bytes completos pela rede (lembrar que as funções *read*/*write* nem sempre lêem/escrevem tudo o que pedimos).
  Um bom sítio para concretizar essas funções é num módulo separado, a ser incluído pelo cliente e servidor, ou no *message-private.h* (concretizando-as no *message.c*).
- Usar a função *signal()* para ignorar sinais do tipo SIGPIPE, lançados quando uma das pontas comunicantes fecha o *socket* de maneira inesperada. Isto deve ser feito tanto no cliente quanto no servidor, evitando que um programa morra quando a outra parte é desligada.

.

Para cada mensagem enviada pelo cliente, deverá haver uma mensagem de resposta correspondente. A tabela seguinte define como deve ser configurada a estrutura message_t para cada uma das invocações feitas pelo cliente e das respostas dadas pelo servidor.

COMANDO CLIENTE	OPCODE PEDIDO	OPCODE RESPOSTA	C_TYPE PEDIDO	CT_TYPE RESPOSTA
PUT	OC_PUT	OC_PUT+1	CT_ENTRY	CT_RESULT
GET	OC_GET	OC_GET+1	CT_KEY	CT_VALUE (chave existe ou não) CT_KEYS (chave = "!")
DEL	OC_DEL	OC_DEL+1	CT_KEY	CT_RESULT
UPDATE	OC_UPDATE	OC_UPDATE+1	CT_ENTRY	CT_RESULT
SIZE	OC_SIZE	OC_SIZE+1	-	CT_RESULT
quit	-	-	-	-

Caso algum dos pedidos não possa ser atendido devido a um erro, o servidor vai retornar {OC_RT_ERROR, CT_RESULT, *errcode*}, onde *errcode* é o código do erro retornado ao executar a operação na tabela do servidor (em geral,-1). Para tal, é necessário adicionar esse novo *opcode*, por exemplo com valor 99, no vosso *message-private.h*.
Note que o caso onde uma chave não é encontrada no *get* não se caracteriza como erro. Quando isso ocorre, o servidor deve retornar de acordo com a tabela acima, mas definindo um `data_t` com `size=0` e `data=NULL`.

6. Entrega

A entrega do projeto 3 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 1/11/2015, até às 22:00hs.

7. 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.