# INE5418 Computação Distribuída

## 2009.1

## Trabalho 1

Parte I: Fundamentos

### 1. Descrição Geral

A parte I (Fundamentos) da disciplina será avaliada através de um trabalho de implementação que esta dividido em 2 projetos, sendo que cada um deles corresponde a partes necessárias para a realização do seguinte trabalho.

O objetivo final é fazer um sistema nos moldes do *Dynamo*, um serviço de tabela contendo pares (chave, valor) (no melhor estilo das implementações da interface *java.util.Map* na API do Java) usado pela *amazon.com* para suportar as diversas aplicações que fazem com que sua infra-estrutura de comércio eletronico funcione [1]. Este nosso sistema será doravante chamado *TyDyn* (*Tyni Dynamo*).

No projeto 1 temos dois objetivos fundamentais:

1.) Construir uma série de funções e estruturas de dados que servirão de base para a implementação das aplicações cliente e servidor a serem construídas no projeto seguinte;

2.) Aprender a lidar com a gestão de memória e a codificação de estruturas em buffers e vice-versa. Consolidando assim um conjunto de competências fundamentais para a construção de sistemas distribuídos eficientes e confiáveis usando a linguagem de programação C [2].

O projeto 1 consiste na implementação de três módulos fundamentais do *TyDyn* : a estrutura de dados onde serão armazenados pares (chave, valor) nos servidores; a codificação e descodificação de *structs* complexas em mensagens; e a leitura dos IPs de um conjunto de servidores a partir de um arquivo de texto. Para cada um destes módulos, é fornecido um arquivo *.h* com os cabeçalhos das funções. As implementações das funções definidas em um arquivos *X.h* devem ser feitas em um arquivo *X.c*, utilizando os algoritmos e métodos que o grupo achar melhor. Os arquivos *.h* apresentados neste documento podem ser obtidos no site da cadeira, se seção dos projetos.

### 2. Estrutura de Dados no Servidor

Um servidor *TyDyn* deve armazenar os dados a ele enviados e suas respectivas chaves. Este armazenamento se dá em uma tabela que oferece operações do tipo *put*, *get*, *remove*, *listkeys*, e *size*.

A estrutura de dados ideal para armazenar o tipo de informação que desejamos e que oferece métodos de pesquisa eficientes é a **tabela *hash*** [3]. O arquivo *tabela.h* que define as estruturas e as funções a serem concretizadas neste módulo é o seguinte.

```
#ifndef _TABELA_H
#define _TABELA_H

/* Funcoes de uma biblioteca de gerenciamento de uma lista. A melhor
 * forma de implementar essa biblioteca é usar uma HASHTABLE
 * para a gerenciamento dos dados.
 */

struct key_t {
      int keysize; /* tamanho da string da chave */
```

```
      char *key; /* string contendo a chave */
};

struct data_t {
      int datasize; /* tamanho do bloco de dados em bytes */
      char *data; /* bloco de dados */
};

/* Funcao para adicionar um elemento na tabela.
 * A função vai copiar a key e os dados num espaco de memoria alocado
 * por malloc().
 * Se a key ja existe, vai substituir essa entrada pelos novos dados.
 * Devolve 0 (ok) ou -1 (out of memory)
 */
int puttab(struct key_t key, struct data_t data);

/* Funcao para obter um elemento da tabela.
 * O primeiro argumento indica a key da entrada da tabela, o
 * segundo elemento é uma struct data_t que contem um ponteiro
 * para um espaço de memoria e seu tamanho.
 * A função vai copiar os dados que correspondem a key para
 * este espaço, e vai atualizar data->datasize para o tamanho
 * do bloco de dados lido.
 * Devolve 0 (ok), -1 (not found), x>0 se o verdadeiro tamanho
 * dos dados é x, mas foram copiados só uma parte (primeiros
 * datasize em bytes) dos dados, por não haver espaço.
 */
int gettab(struct key_t key, struct data_t *data);

/* Função para remover um elemento da tabela. Vai liberar
 * toda a memoria alocada na respectiva operação puttab().
 * Devolve: 0 (ok), -1 (key not found)
 */
int removetab(struct key_t key);

/* Devolve número de elementos da tabela.
 */
int tabsize();

/* Devolve um array com a cópia de todas as keys da tabela.
 */
struct key_t *gettabkeys();

/* Libera a memoria alocada por getkeys()
 */
void freetabkeys(struct key_t *keys);


#endif
```

É importante ressaltar que todas as funções devem manipular uma tabela *hash* declarada como uma variável global (usar palavra reservada *static*) do programa e que a definição da função da *hash* usada para mapear chaves em posições na tabela é também parte do trabalho.

## 3. *Marshaling* e *Unmarshaling* de Mensagens

Esta parte do projeto consiste em transformar uma estrutura de dados complexa em um *buffer* contendo seus dados, de tal forma que este *buffer* esteja pronto a ser enviado pela rede. O arquivo *mensagem.h* define as estruturas e as funções a serem implementadas.

```
#ifndef _SD_MENSAGEM_H
#define _SD_MENSAGEM_H

#include "tabela.h"

/* Estrutura que representa uma mensagem a ser enviada */
struct message_t {
      short opcode;
      struct key_t key;
      struct data_t data;
};

/* estrutura que representa um produto, com nome e preco. */
struct product_t {
      int size_n;
      char *name;
      int price;
};

/* formato da mensagem serialisado:
 * OPCODE  KEYSIZE KEY             DATASIZE DATA
 * [2byte] [4byte] [KEYSIZE bytes] [4 byte] [DATASIZE bytes]
 */

/* Encoding struct -> char[], returna o tamanho do buffer alocado, a
 * ser retornada pelo parâmetro buffer.
 */
int message_to_buffer(struct message_t *msg, char **buffer);
int product_to_buffer(struct product_t *prod, char **buffer);

/* Liberar memoria alocada por XX_to_buffer */
void cleanup_buffer(char **buffer);


/* Decoding char[] -> struct, retorna a struct devidamente alocada */
struct message_t *buffer_to_message(int buflen, char *buffer);
struct product_t *buffer_to_product(int buflen, char *buffer);

/* Liberar memoria alocada por buffer_to_message */
void cleanup_message(struct message_t *message);

/* Liberar memoria alocada por buffer_to_product */
void cleanup_product(struct product_t *product);

#endif
```

Os campos de dados de 32 e de 16 bits devem ser escritos no buffer sempre em formato de rede (*big endian*) utilizando para o efeito as funções de conversão `htonl`, `htons`, `ntohl` e `ntohs` (ver respectivas *Manual Pages* [4]).

## 4. Leitura de Um arquivo de Configuração

Esta parte do trabalho consiste apenas na leitura de um arquivo que contém uma lista de endereços IPs de servidores (um por linha) e colocar as *strings* com estes endereços em um *array* de *strings* (tipo *char***). O arquivo *config.h* contendo a especificação das funções a serem implementadas é o seguinte.

```
#ifndef _CONFIG_H
#define _CONFIG_H
```

```
/* config.h
 *
 * Funções para ler um arquivo de configuração que contem enderecos
 * IP de servidores, formato texto com uma linha por servidor.
 */

/* MAX_SERVERS: Numero maximo de servidores. Se o arquivo
 * tiver mais linhas, deve-se ignorar as linhas a mais.
 */
#define MAX_SERVERS 20

/* Função para ler o arquivo indicado pelo primeiro
 * argumento. O segundo argumento deve ser um array
 * com espaço para guardar MAX_SERVERS ponteiros.
 * A função vai ler o arquivo e preencher o array
 * com ponteiros para strings com os endereços.
 * (strings terminado por \0, mas sem \n no final!)
 * A memoria para estes strings vai ser alocada pela funcao.
 * Devolve 0 (OK), -1 (file read error), -2 (out of memory)
 */
int read_server_list(char *config_file, char **servers);


/* Liberar memoria alocada numa chama de de read_server_list
 * com servers no segundo argumento. So deve ser chamada apos
 * executar read_server_list com exito.
 */
void cleanup_server_list(char **servers);

#endif
```

## 5. Entrega

- O trabalho pode ser feito em grupo (até 3 alunos).
- Os projetos deverão ser submetidos até a sua data de entrega, via e-mail (fernando@inf.ufsc.br), contendo :

1. Subject do e-mail : <ine5418+projeto> Ex.: ine5418projeto1, ine5418projeto2, ...
2. o arquivo com código fonte+ .h (em anexo),
3. descrição do trabalho, nome do(s) autor(es) e procedimentos de compilação/execução.

   A identificação dos arquivos deverá seguir o seguinte padrão, <trabalho + login [-login2] >, por exemplo: proj*1_fernando.c* , proj*1_fernando.h* .

- **O prazo de entrega para o projeto 1 é dia 01/04/2009 até as 18:00hs**.

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

[3] Wikipedia. *Hash Table*. http://en.wikipedia.org/wiki/Hash_table. Consultado em 16 de Março de 2009.

[4] Linux Man pages. *htonl, etc…* http://www.linuxmanpages.com/man3/htonl.3.php. Consultado em 16 de Março de 2009.