Ciências ULisboa | Informática

2021/22

Sistemas Distribuídos

Projeto 3

1. Descrição Geral

A componente teórico-prática da disciplina de sistemas distribuídos consiste no desenvolvimento de quatro projetos, utilizando a linguagem de programação C [4], 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 (nos moldes da interface *java.util.Map* da API Java) similar ao utilizado pela *Amazon* para dar suporte aos seus serviços Web [1]. Neste sentido, a estrutura de dados utilizada para armazenar esta informação é uma **lista encadeada simples** [2] e uma **tabela *hash*** [3], dada a sua elevada eficiência ao nível da pesquisa.

No Projeto 1 foram definidas estruturas de dados e implementadas várias funções para lidar com a manipulação dos dados que vão ser armazenados na tabela, bem como para gerir uma *tabela hash* local que suporte um subconjunto dos serviços definidos pela tabela *hash*. No Projeto 2 implementaram-se as funções necessárias para serializar e de-serializar estruturas complexas usando Protocol Buffer, um servidor concretizando a *tabela hash*, e um cliente com uma interface de gestão do conteúdo da *tabela hash*.

No Projeto 3 iremos criar um sistema concorrente que aceita e processa pedidos de múltiplos clientes em simultâneo através do uso de múltiplas *threads*. Mais concretamente, vai ser preciso:

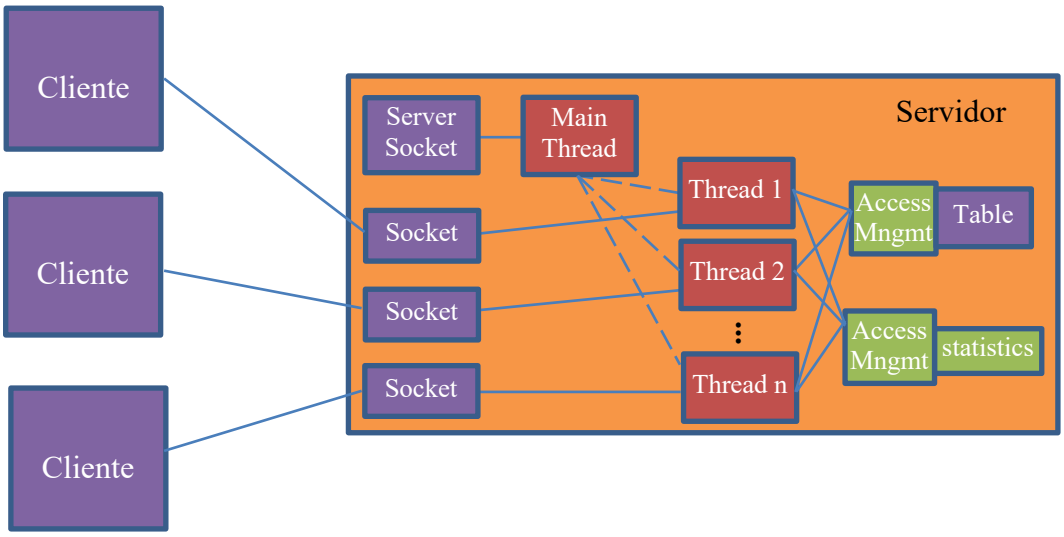
- Adaptar o servidor de modo a que este suporte pedidos de múltiplos clientes ligados em simultâneo. Para cada pedido de ligação de cliente deverá ser criada uma *thread*, que irá tratar e atender exclusivamente os pedidos daquele cliente, até que este se desligue. A *thread* deverá ser criada no servidor no momento da aceitação do pedido de ligação do cliente;
- Implementar no servidor uma estrutura *statistics* que deverá armazenar as seguintes estatísticas sobre o servidor:
  - Número de vezes que cada operação na tabela foi executada no servidor;
  - Tempo médio de atendimento de pedidos dos clientes.
- Implementar uma operação *stats*, que permite que o cliente obtenha as estatísticas registadas na estrutura *statistics* do servidor.
- Adaptar o servidor para ter as seguintes *threads*:
  - a *thread* principal do programa, que fica responsável por aceitar novas ligações de clientes e por criar *threads* secundárias que processarão os pedidos dos clientes;
  - *threads* secundárias, lançadas pela *thread* principal, em que cada *thread* secundária irá tratar e atender exclusivamente os pedidos de um cliente.
- Gestão da concorrência das *threads* secundárias no acesso à tabela e à estrutura *statistics* do servidor, através do uso de mecanismos de gestão da concorrência.

Como nos projetos anteriores, espera-se uma grande fiabilidade por parte do servidor e cliente, portanto não podem existir condições de erro não verificadas ou gestão de memória ineficiente a fim de evitar que estes sofram um *crash*.

2. Descrição Detalhada

O objetivo específico do projeto 3 é desenvolver uma aplicação do tipo cliente-servidor capaz de suportar múltiplos clientes de forma síncrona. Para tal, para além de aproveitarem o código desenvolvido nos projetos 1 e 2, os alunos devem fazer uso de novas técnicas ensinadas nas aulas, incluindo a biblioteca de *threads* `pthreads` para separar a execução de pedidos de clientes, e técnicas de *gestão da concorrência* tais como *mutexes* e *condições* para garantir a sincronização de *threads* no acesso a variáveis ou estruturas de dados na memória partilhada. Devem também implementar uma estrutura de dados simples para armazenar estatísticas do servidor.

A figura abaixo ilustra o novo modelo de comunicações que será usado no projeto 3. Atenção que este modelo abstrai os detalhes de comunicação implementados no projeto 2, ou seja, não mostra os módulos *stub*, *skeleton* e network apenas por simplificação na apresentação. A cor roxa representa o que já foi feito nos projetos 1 e 2, e as cores verde e vermelho representam o que é preciso fazer neste projeto 3.



2.1. Servidor com múltiplas threads

Nesta secção apresenta-se uma breve descrição do novo código a ser desenvolvido na função `network_main_loop(int listening_socket)` do servidor, mais especificamente no *network_server*, bem como do código a ser executado pela *thread* secundária. Assume-se que o `listening_socket` recebido como argumento já foi preparado para receção de pedidos de ligação num determinado porto, tal como especificado no projeto 2.

```
/*
 * Esboço do algoritmo a ser implementado na função network_main_loop
 */

while (1) {
      connsockfd = accept(listening_socket);
      cria nova thread secundária passando-lhe connsockfd;
}

/*
 *  Thread secundária de atendimento do cliente
*/

      while ((message = network_receive(connsockfd)) !=NULL) {      /* cliente não fechou conexão */
            invoke(message);                                        /* Executa pedido contido em message */
            network_send(connsockfd,message);                       /* Envia resposta contida em message */
      }
      close(connsockfd);
      termina a thread secundária;
```

De notar que apenas se apresenta a ideia geral de como deve ser a função `network_main_loop` do servidor e a *thread* secundária. Cabe aos alunos traduzir essa lógica para código C (ou inventar outro algoritmo).

2.2. Servidor com atendimento síncrono dos pedidos

O servidor deve processar os pedidos dos clientes com *threads* exclusivas para cada ligação, ou seja, seguindo um modelo *thread per client*. Todos os pedidos enviados pelos clientes são atendidos de forma síncrona, o que implica que os clientes têm de ficar bloqueados à espera da resposta.

2.2.1 Resposta a pedidos de estatísticas

O servidor passa a guardar internamente uma estrutura *statistics* que deverá armazenar estatísticas sobre os operações realizadas no servidor (nomeadamente: o número de vezes que cada operação foi executada no servidor; e o tempo médio de atendimento de pedidos dos clientes). Sempre que uma *thread* processa uma operação, o contador da respetiva operação é incrementado e o tempo médio de atendimento dos pedidos é recalculado. Assim, a estrutura *statistics* tem de ser partilhada por todas as *threads*,

Adicionalmente, os alunos devem implementar a nova operação *stats*, através da qual o cliente deve ser capaz de obter as estatísticas do servidor. Segue uma apresentação do formato das mensagens referentes à operação *stats*:

COMANDO UTILIZADOR	**MENSAGEM DE PEDIDO**	**MENSAGEM DE RESPOSTA**
stats	OP_STATS CT_NONE	OP_STATS+1 CT_RESULT <result> OP_ERROR CT_NONE <none>

Tabela 1: Novo pedido *stats*

Assim como o novo *opcode*:

```
/* Define os possíveis opcodes da mensagem */
...
OP_STATS          70
...
```

Não esquecer de adicionar o método *stats* ao *client_stub.c/h*:

```
#ifndef _CLIENT_STUB_H
#define _CLIENT_STUB_H

...

/* Obtém as estatísticas do servidor.*/


struct statistics *rtable_stats(struct rtable_t *rtable);

#endif
```

Não esquecer também de incluir o tratamento da operação *stats* no método `invoke` do table_skel.c.

2.2.2 Servidor com Threads e gestão da concorrência

Para processar os pedidos em paralelo, o servidor (nomeadamente, o *network_server* na função *network_main_loop*) deve lançar novas *threads* secundárias (uma para cada ligação de cliente), através da API de *threads* do UNIX (`pthreads`). A *thread* principal deve estar sempre à espera de novos pedidos de ligação. Cada nova *thread* secundária deve processar os pedidos que forem recebidos por meio da respetiva ligação com o cliente.

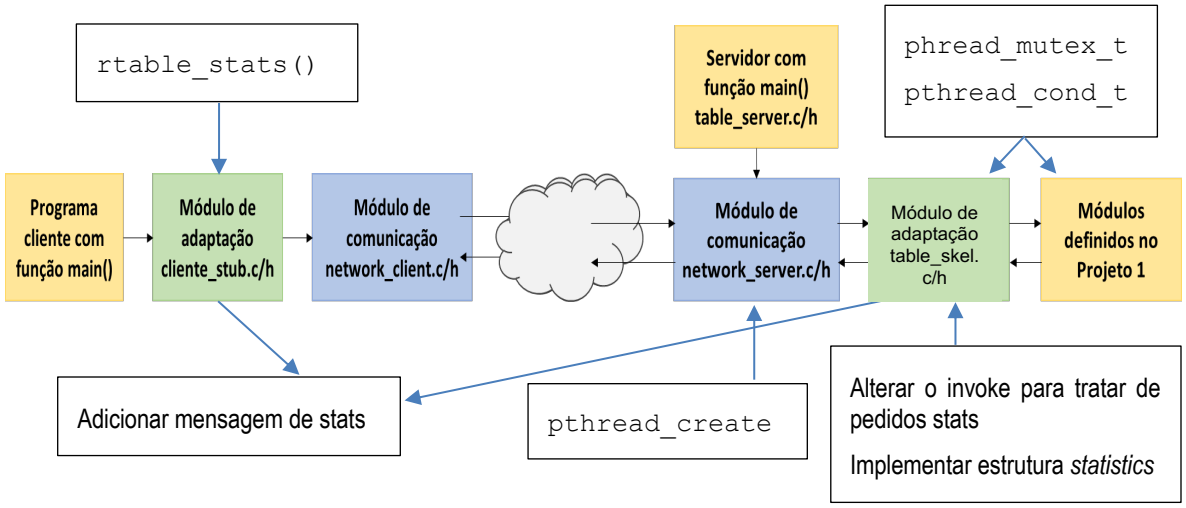
Isto significa que há duas estruturas que poderão ser modificadas concorrentemente pelas *threads* secundárias: a estrutura *statistics* e a tabela *hash*. Assim, para garantir que o acesso a estas estruturas é feito de forma ordenada, que os resultados das operações sobre as mesmas são corretos, e que não existe o risco de as estruturas ficarem corrompidas, torna-se necessário utilizar mecanismos de gestão da concorrência.

Existem dois mecanismos que podem ser usados para concretizar a gestão da concorrência: *locks* (`pthread_mutex_t`) e variáveis condicionais (`pthread_cond_t`). A forma como são utilizados é da responsabilidade dos alunos. Contudo, os alunos devem ter em conta o seguinte:

- Relativamente à tabela *hash,* dado que existem várias operações que os clientes podem realizar, deve-se permitir a execução concorrente tanto quanto possível, evitando bloquear operações que possam ser executadas sem comprometer a correção do sistema.
- Quanto à estrutura *statistics*, dado que apenas existe uma operação de escrita (para atualizar um contador, qualquer que seja, e atualizar o tempo de atendimento) e uma operação de leitura (para obter todos os dados armazenados na estrutura), deve ser implementada uma solução de sincronização dos acessos do tipo "*single writer / multiple readers*". Isto significa que as operações de escrita são sempre feitas em exclusividade de acesso, mas podem ser feitas várias operações de leitura em simultâneo.

3. Sumário de Alterações

Esta secção apresenta um sumário de onde deve ser feita cada alteração, dada a estrutura de ficheiros definida no projeto 2.



4. Makefile

Os alunos deverão manter o `Makefile` usado no Projeto 2, atualizando-o para compilar novo código se necessário.

5. Entrega

A entrega do projeto 3 tem de ser feita de acordo com as seguintes regras:

1. Colocar todos os ficheiros do projeto, bem como o ficheiro README mencionado abaixo, num ficheiro com compressão no formato ZIP. O nome do ficheiro será **`grupoXX-projeto3.zip`** (XX é o número do grupo).
2. Submeter o ficheiro **`grupoXX-projeto3.zip`** na página da disciplina no Moodle da FCUL, utilizando a atividade disponibilizada para tal. Apenas um dos elementos do grupo deve submeter e todos os elementos têm de confirmar a submissão.

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 podem incluir informações que julguem necessárias (e.g., limitações na implementação);
- diretorias adicionais, nomeadamente:
  - include: para armazenar os ficheiros .h;
  - source: para armazenar os ficheiros .c;
  - object: para armazenar os ficheiros objeto;
  - lib: para armazenar bibliotecas;
  - binary: para armazenar os ficheiros executáveis.
- um ficheiro `Makefile` que satisfaça o especificado na Secção 4. Não devem ser incluídos no ficheiro ZIP os ficheiros objeto (.o) ou executáveis. Quaisquer outros ficheiros (por exemplo, de teste) também não deverão ser incluídos no ficheiro ZIP.

Na entrega do trabalho, é ainda necessário ter em conta que:

- **Se não for incluído um `Makefile`, se o mesmo não satisfizer os requisitos indicados, ou se houver erros de compilação (isto é, se não forem criados os ficheiros objeto e executáveis), o trabalho é considerado nulo**. Na página da disciplina, no Moodle, 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 um cabeçalho com três ou quatro linhas de comentários a dizer o número do grupo e o nome e número dos seus elementos.
- Os programas são testados no ambiente dos laboratórios de aulas, pelo que se recomenda que os alunos testem os seus programas nesse ambiente.

O prazo de entrega é dia 22/11/2021 até às 23:59hs.

Após esta data, a submissão do trabalho através do Moodle deixará de ser permitida. Também, cada grupo ficará sem acesso de escrita à diretoria de entrega.

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] Wikipedia. Linked List. https://en.wikipedia.org/wiki/Linked_list.

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

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