Base-de-Dados em Anel com Cordas

Redes de Computadores e Internet
2º Semestre 2021/2022
Projeto de Laboratório, versão 2

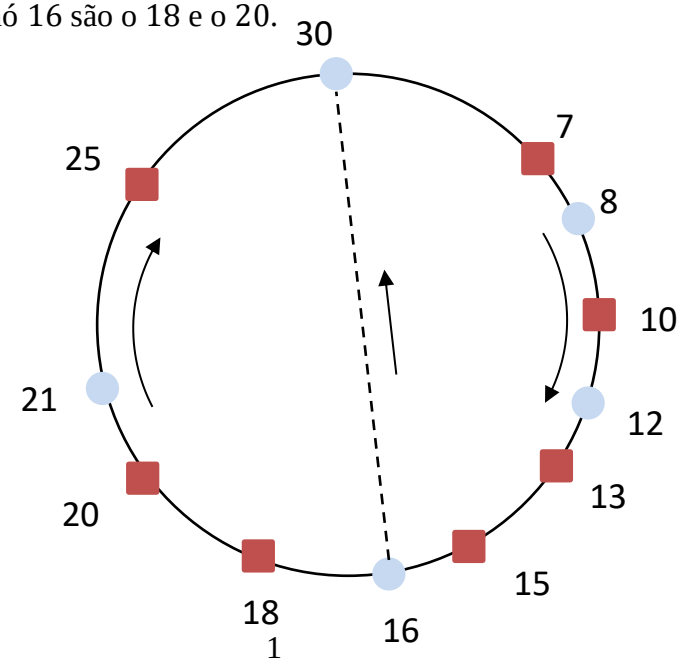
1. Introdução

Imagine uma base-de-dados composta por (muitos) objetos particionados por (poucos) nós. Cada objeto está guardado em um só nó. São três as operações básicas nesta base-de-dados: um nó pretende saber em que nó está guardado determinado objeto; um nó pretende entrar na base-de-dados, guardando em si alguns dos objetos que estavam guardados noutros nós; e um nó pretende sair da base-de-dados, atribuindo os seus objetos a outros nós.

1.1 Anel com cordas

A organização da base-de-dados em anel permitirá uma entrada e saída eficiente dos nós. Considere-se o conjunto dos primeiros N números naturais ordenados em anel, isto é, tal que ao número $N-1$ se segue o número 0, sendo que N é (muito) superior ao número de objetos adicionado ao número de nós. Cada um destes números é chamado de *chave*. Cada objeto tem uma chave única e cada nó tem, também, uma chave única. Cada nó tem um *sucessor* no anel que é o nó que se lhe segue no anel e um *predecessor* que é o nó que o tem como sucessor.

Os objetos cujas chaves estão entre a chave do nó e a do seu sucessor estão guardados no nó. Na figura, $N = 32$, os círculos azuis representam nós e os quadrados verdes representam objetos. O sucessor do nó 16 é o nó 21 e o seu predecessor é o nó 12. Os objetos guardados no nó 16 são o 18 e o 20.



Quando um nó entra, ele guarda em si os objetos com chave entre a sua e a do seu sucessor, retirando-os ao anterior predecessor deste. Quando um nó sai, os seus objetos ficam guardados no seu anterior predecessor. Na figura, se um nó 19 entrar na base-de-dados, então ele passa a guardar o objeto 20, ficando o nó 16 a guardar apenas o objeto 18. Se, ao invés, o nó 16 sair da base-de-dados, então o nó 12 passa a guardar, além dos objetos 13 e 15, também os objetos 18 e 20.

Resta saber como é que um nó descobre qual o nó que guarda um objeto a partir da chave deste. O nó lança uma mensagem de procura pela chave em torno do anel. A mensagem para no nó que guarda o objeto. Este nó lança então uma mensagem de resposta à procura que completa uma volta em torno do anel. Na figura, se o nó 21 procurar o objeto 10, então ele lança uma mensagem que atravessa o nó 30 e para no nó 8 (que guarda o objeto 10). O nó 8 responde com uma mensagem que atravessa os nós 12 e 16 antes de chegar ao nó 21 (que lançou a procura).

Tal como descrito, uma procura e a correspondente resposta dão uma volta completa por todos os nós no anel. Para tornar a descoberta mais eficiente, um nó pode escolher um outro nó como *atalho*, criando uma *corda* no anel. Oportunamente, o nó pode expedir procuras e respostas para o seu atalho em vez de para o seu sucessor. Desta feita, uma procura e correspondente resposta circulam por um sub-anel contendo uma ou mais cordas. Na figura, se o nó 12 pesquisar o objeto 10, então ele lança uma mensagem que é primeiro recebida no nó 16. Este nó tem um atalho para o nó 30 que está mais perto do objeto 10 do que o seu nó sucessor 21. O nó 16 retransmite a mensagem para o nó 30 que a retransmite para o nó 8 parando então a procura. A resposta é depois entregue pelo nó 8 ao nó 12.

2. Operações no anel com cordas

Consideramos quatro operações no anel com cordas deste projeto:

- a procura de uma chave (em vez de um objeto);
- a entrada de um nó sabendo a sua posição no anel;
- a saída de um nó;
- a entrada de um nó não sabendo a sua posição no anel.

2.1 Distâncias

Seja N o número de chaves. A *distância* da chave k à chave l é $d_N(k,l) = (l-k) \bmod N$, ou seja, é o resto da divisão de $(l-k)$ por N. Note que a distância da chave k à chave l não é igual à distância da chave l à chave k; na verdade, $d_N(k,l) + d_N(l,k) = N$.

O sucessor do nó i, $s(i)$, é o nó que está à menor distância de i. A chave k *pertence* ao nó i se a distância de i a k for menor do que a distância de $s(i)$ a k. O predecessor do nó i, $p(i)$, é o nó que tem i como sucessor, $s(p(i)) = p(s(i)) = i$. O nó i tem, no máximo, um atalho $a(i)$ que é um qualquer outro nó do anel.

2.3 Pesquisa de uma chave

Um nó i que procure saber qual o nó a que pertence uma chave k (que não pertence a ele próprio) inicia uma pesquisa por k ao longo do anel ou sub-anel. O nó a que pertence k responderá com uma pesquisa por i completando assim uma volta ao anel ou sub-anel.

Considere um nó j encarregado de pesquisar k por iniciativa do nó i.

1. Se a distância de j a k for menor do que a distância de $s(j)$ a k, $d_N(j,k) < d_N(s(j),k)$, então k pertence a j e a pesquisa termina.
2. Pelo contrário, se a distância de j a k for maior do que a distância de $s(j)$ a k, $d_N(j,k) > d_N(s(j),k)$, então j delega a pesquisa:
   2.1. em $a(j)$, se $a(j)$ existir e a distância de $a(j)$ a k for menor do que a distância de $s(j)$ a k, $d_N(a(j),k) < d_N(s(j),k)$;
   2.2. em $s(j)$, no caso contrário.

2.4 Entrada de um nó

A entrada de um novo nó no anel pressupõe que este tenha conhecimento de qual será o seu predecessor. Seja i o nó entrante e j o seu futuro predecessor no anel. Quando o nó i entra, ele tem que inicializar o seu predecessor para j e o seu sucessor para $s(j)$. Por outro lado, o nó j tem que atualizar o seu sucessor para i e o nó $s(j)$ tem que atualizar o seu predecessor para i. Concretamente, a entrada do nó i pede as seguintes ações.

1. O nó i inicializa o seu predecessor para j e entra no anel dando-se a conhecer a j.
2. O nó j atualiza o seu sucessor para i, informa o seu antigo sucessor $s(j)$ sobre o nó i e abre o anel com $s(j)$.
3. O nó $s(j)$ atualiza o seu predecessor para i, fecha o anel com o nó i e dá-se a conhecer a i.
4. O nó i atualiza o seu sucessor para $s(j)$.

2.5 Saída de um nó

Quando o nó i sai, o sucessor $s(i)$ tem que atualizar o seu predecessor para $p(i)$ e o predecessor $p(i)$ tem que atualizar o seu sucessor para $s(i)$. Concretamente, a saída do nó i pede as seguintes ações.

1. O nó i informa o seu sucessor $s(i)$ sobre o seu predecessor $p(i)$ e sai do anel.
2. O nó $s(i)$ atualiza o seu predecessor para $p(i)$, fecha o anel com este nó e dá-se a conhecer a $p(i)$.
3. O nó $p(i)$ atualiza o seu sucessor para $s(i)$.

2.6 Descoberta da posição de um nó no anel

E se um nó entrante com chave i não souber qual é o seu predecessor no anel, ao invés, sabendo apenas da existência de um qualquer nó do anel? Neste caso, o nó entrante pede ao nó no anel que conhece que faça uma procura pela chave i. A resposta a esta procura produz o nó que virá a ser o predecessor de i.

3. Especificação

Cada grupo de dois alunos deve concretizar a aplicação **`ring`** correspondendo a um nó num anel com $N = 32$ chaves. O anel é concretizado em sessões TCP que formam uma *rede de sobreposição* à rede de computadores que a suporta. A comunicação entre um nó e o seu sucessor e predecessor é por TCP. A comunicação entre um nó e o seu atalho é por UDP tal como é a comunicação entre um nó entrante e um nó do anel que aquele interroga com o objetivo de determinar o seu predecessor. A aplicação **`ring`** é composta pelos elementos seguintes:

- comando de invocação da aplicação;
- interface de utilizador;
- protocolo de pesquisa de uma chave;
- protocolo para entrada e saída de um nó;
- protocolo para a descoberta da posição de um nó no anel.

3.1 Comando de invocação da aplicação

A aplicação **`ring`** é invocada com o comando

`ring i i.IP i.port`

em que **`i`**, **`i.IP`** e **`i.port`** são, respetivamente, a chave, o endereço IP e o porto de um nó. Em resultado desta invocação, a aplicação disponibiliza a interface de utilizador especificada de seguida.

3.2 Interface de utilizador

A interface de utilizador consiste nos seguintes comandos, os quais devem ser abreviados pelas suas letras iniciais.

- **`new`**
  Criação de um anel contendo apenas o nó.
- **`bentry boot boot.IP boot.port`**

Entrada do nó no anel ao qual pertence o nó ***boot*** com endereço IP ***boot.IP*** e porto ***boot.port***. Embora apareça primeiro nesta lista, este será o último comando a ser implementado.

- **pentry** ***pred pred.IP pred.port***
  Entrada do nó no anel sabendo que o seu predecessor será o nó ***pred*** com endereço IP ***pred.IP*** e porto ***pred.port***. Com este comando, um nó pode entrar no anel sem que a funcionalidade de procura de chaves esteja ativa.
- **chord** ***i i.IP i.port***
  Criação de um atalho para o nó ***i*** com endereço IP ***i.IP*** e porto ***i.port***.
- **echord**
  Eliminação do atalho.
- **show**
  Mostra o estado do nó, consistindo em: (i) a sua chave, endereço IP e porto; (ii) a chave, endereço IP e porto do seu sucessor; (iii) a chave, endereço IP e porto do seu predecessor; e, por fim, (iv) a chave, endereço IP e porto do seu atalho, se houver.
- **find** ***k***
  Procura da chave ***k***, retornando a chave, o endereço IP e o porto do nó à qual a chave pertence.
- **leave**
  Saída do nó do anel.
- **exit**
  Fecho da aplicação.

3.3 Pesquisa de uma chave

Na pesquisa de uma chave são usadas quatro mensagens protocolares semelhantes: elas diferem consoante a pesquisa está associada a uma procura ou a uma resposta e consoante a pesquisa é enviada por TCP ou por UDP. As mensagens enviadas por TCP têm que ser terminadas, enquanto que as mensagens enviadas por UDP não têm que ser terminadas, mas devem ser confirmadas.

- <**FND** ***k n i i.IP i.port*****\n** >
  Um nó delega no seu sucessor a procura da chave ***k*** que foi iniciada pelo nó ***i*** com número de sequência ***n***, sendo o endereço IP deste nó ***i.IP*** e o seu porto ***i.port***. O número de sequência ***n*** situa-se entre 0 e 99. Os campos são separados por espaços brancos simples e a mensagem é terminada como o caráter **\n**.
- <**RSP** ***k n i i.IP i.port*****\n** >
  Um nó delega no seu sucessor um resposta destinada ao nó com chave ***k*** que foi iniciada no nó ***i***, o qual que terá copiado o número de sequência ***n*** da procura que deu origem à resposta, sendo o endereço IP deste nó ***i.IP*** e o seu porto ***i.port***. (O número de sequência da procura e da resposta correspondente serão, portanto, o mesmo.)

- <**FND** ***k n i i.IP i.port*** >
  Um nó delega no seu atalho a procura da chave ***k*** que foi iniciada pelo nó ***i*** com número de sequência ***n***, sendo o endereço IP deste nó ***i.IP*** e o seu porto ***i.port***.
  <**RSP** ***k n i i.IP i.port*** >
  Um nó delega no seu atalho um resposta destinada ao nó com chave ***k*** que foi iniciada no nó ***i***, o qual que terá copiado o número de sequência ***n*** da procura que deu origem à resposta, sendo o endereço IP deste nó ***i.IP*** e o seu porto ***i.port***.
- <**ACK** >
  Confirmação de qualquer uma das duas mensagens anteriores.

3.4 Saída e entrada de um nó

Na entrada e saída de um nó são usadas duas mensagens protocolares enviadas sobre TCP.

- <**PRED** ***pred pred.IP pred.port\n*** >
  Um nó informa o seu sucessor atual sobre o novo predecessor deste ***pred*** que tem endereço IP ***pred.IP*** e porto ***pred.port***.
- <**SELF** ***i i.IP i.port\n*** >
  O nó ***i*** dá-se a conhecer ao seu predecessor com o seu endereço IP ***i.IP*** e porto ***i.port***.

3.5 Descoberta da posição de um nó no anel

A descoberta da posição de um nó no anel requer duas mensagens protocolares enviadas sobre UDP e respetivas confirmações.

- <**EFND** ***i*** >
  Um nó entrante solicita a um nó no anel que descubra a sua posição nesse anel, retornando a chave, endereço IP e porto do seu futuro predecessor.
- <**EPRED** ***pred pred.IP pred.port*** >
  Um nó do anel responde a um nó entrante com a chave do futuro predecessor deste ***pred*** que tem endereço IP ***pred.IP*** porto ***pred.port***
- <**ACK** >
  Confirmação de qualquer uma das duas mensagens anteriores.

4. Desenvolvimento

Cada grupo de alunos deve adquirir a destreza necessária sobre programação em redes para realizar a aplicação proposta.

4.1 Etapas recomendadas

As etapas seguintes são recomendadas na concretização do projeto. Os números em parêntese retos são a duração de referência para cada etapa.

1. Criação de um anel com os comandos ***new*** e ***pentry***. Comece com dois nós e duas sessões TCP entre eles. [1 semana]
2. Mostra do estado, comando ***show***. [0 semanas]
3. Saída de um nó do anel, comando ***leave***. [1 semana]
4. Pesquisa de uma chave, comando ***find***. Posteriormente, introdução de atalhos, comando ***chord***, e nova pesquisa de uma chave, comando ***find***. [1 semana]
5. Entrada de um nó no anel apenas com informação sobre um qualquer nó do anel, comando ***bentry***. [1 semana]

4.2 Chamadas de sistema

O código da aplicação fará uso das seguintes chamadas de sistema:

- leitura de informação do utilizador para a aplicação: `fgets()`;
- conversão entre *strings* e tipos de dados: `sscanf()`, `sprintf()`;
- gestão de um cliente UDP: `socket()`, `close()`;
- gestão de um servidor UDP: `socket()`, `bind()`, `close()`;
- comunicação UDP: `sendto()`, `recvfrom()`;
- gestão de um cliente TCP: `socket()`, `connect()`, `close()`;
- gestão de um servidor TCP: `socket()`, `bind()`, `listen()`, `accept()`, `close()`;
- comunicação TCP: `write()`, `read()`;
- multiplexagem de informação: `select()`.

4.3 Depuração

Faça uso das estratégias seguintes que permitem uma boa depuração do código:

- comente o código à medida que o desenvolve;
- use um depurador, `gdb` ou `xgdb`, para inspecionar a execução sequencial do código num nó;
- use o comando `nc` (`netcat`) para testar a comunicação com um cliente ou com um servidor, quer seja TCP ou UDP;
- corra o analisador de protocolos Wireshark para visualizar todas as mensagens trocadas com um nó e seus conteúdos;
- tente interoperar um nó implementado pelo seu grupo com um nó implementado por outro grupo (isto é possível porque o formato das mensagens é o mesmo para todos).

Quer os clientes quer os servidores devem terminar graciosamente, pelo menos nas seguintes situações de falha:

- mensagens mal formatadas;
- sessão TCP fechada abruptamente;
- erro nas chamadas de sistema.

5. Entrega do Projecto

Cada grupo deve submeter no sistema fénix: (i) código fonte da aplicação **`ring`**; (ii) a respetiva `makefile`; e (iii) uma ficha de auto-avaliação preenchida que será disponibilizada mais tarde. A execução do comando `make` deverá produzir o executável **`ring`** sem quaisquer erros ou avisos. O código será testado apenas no ambiente de desenvolvimento do laboratório.

A data limite de submissão do projeto é terça-feira, dia é 12 de abril, às 23:59 (imediatamente antes das férias da Páscoa).

6. Bibliografia

- José Sanguino, A Quick Guide to Networking Software, 5ª edição, 2020.
- W. Richard Stevens, Unix Network Programming: Networking APIs: Sockets and XTI (Volume 1), 2ª edição, Prentice-Hall PTR, 1998, ISBN 0-13-490012-X, capítulo 5.
- Michael J. Donahoo, Kenneth L. Calvert, TCP/IP Sockets in C: Practical Guide for Programmers, Morgan Kaufmann, ISBN 1558608265, 2000.
- Manual on-line, comando `man`.