Trabalho Prático 2
Servidor de mensagens instantâneas

14 de outubro de 2015

1 Descrição

O serviço de mensagens instantâneas permite o envio e recebimento de mensagens de texto em tempo real entre dois ou mais usuários. Este tipo de serviço se popularizou em meados dos anos 2000 através do mensageiro ICQ[1] e posteriormente com o MSN[2]. Atualmente, aplicativos como *gtalk*, *facebook mensenger* e *whatsapp* já estão entre os meios de comunicação mais usados, superando o uso de SMS e e-mails.

Uma das possíveis arquiteturas, de uma aplicação que oferece serviço de mensagens instantâneas via web, é o modelo cliente-servidor. Neste modelo, as mensagens são enviadas a um servidor que realiza algum tipo de processamento e as envia ao destinatário.

Por se tratar de uma aplicação distribuída com múltiplos usuários, pode ocorrer de uma ou mais mensagens chegarem ao mesmo tempo no servidor. Neste caso, o servidor deve possuir alguma política que determine qual destas mensagens será enviada primeiramente.

Outro problema que pode acontecer, é o de algumas mensagens chegarem fora de ordem devido a problemas ou gargalos na comunicação entre o cliente o servidor. Um sistema de mensagens instantâneas deve garantir a ordem de entrega das mensagens pois, caso contrário, a conversa entre dois usuários poderia ficar sem sentido.

2 O que deve ser feito

Neste trabalho vocês devem implementar um simulador de um servidor de mensagens instantâneas via web. Este servidor deve permitir que pares de usuários troquem mensagens entre si. Antes de detalhar a lógica a ser implementada, algumas considerações:

- Para simplificar, não faremos distinção da direção de comunicação entre um mesmo par de usuários. Ou seja, supondo a existência de dois usuários, A e B, só estamos interessados em saber se uma menagem foi trocada entre A e B, não queremos saber se foi uma mensagem enviada de A para B ou de B para A.

[1] https://icq.com/
[2] https://www.msn.com/

- Cada mensagem será composta por 3 campos: identificador do par de usuários que está trocando a mensagem, identificador de ordem da mensagem, conteúdo da mensagem. O identificador de ordem da mensagem, informa se esta é a primeira, segunda, terceira, ... ou a n-ésima mensagem trocada por aquela par de usuários. Esta informação será necessária para garantir a ordem de entrega das mensagens.

Por ser um trabalho de AEDSII e não de Redes de Computadores, só será necessário implementar as estruturas de armazenamento das mensagens e a política que define a ordem de envio das mensagens. O funcionamento básico da comunicação será o seguinte:

- As mensagens são entregues ao servidor em lotes (conjuntos com uma ou mais mensagens);
- Durante o recebimento de cada lote, o servidor armazena estas mensagens em listas de esperas para que possam ser enviadas em uma próxima etapa. Cada par de usuários que está se comunicando possui uma lista de espera própria. Assim, cada mensagem do lote será armazenada na lista de espera referente ao par de usuários que está trocando aquela mensagem;
- Após o recebimento completo de cada lote, as mensagens estão prontas para serem enviadas aos destinatários. Neste momento, o servidor possui várias listas de esperas com várias mensagens em cada lista. Como mencionado, as mensagens podem chegar fora de ordem ao servidor mas devem ser entregues em ordem aos destinatários. Assim, cada lista de espera deve estar ordenada crescentemente de acordo com o identificador de ordem de suas mensagens. Note que não é necessário ordenar a lista, basta inserir as mensagens de maneira ordenada.
- O servidor irá agora, enviar todas as mensagens das listas de espera. Nesta simulação as mensagens serão enviadas sequencialmente. Para definir a ordem de envio deve ser respeitada a seguinte política de prioridade: O par de usuários que trocou menos mensagens até o momento terá prioridade sobre os outros. Em caso de empate entre dois ou mais pares, o par com menor identificador terá prioridade. Note que a prioridade é sobre o par e a cada envio de mensagem estas prioridades mudam.
- Cada mensagem ao ser enviada, deve ser removida de sua respectiva lista de espera.
- Após o envio de todas as mensagens nas listas de espera, o servidor receberá um novo lote de mensagens e o processo se repete. Note que os contadores de mensagens trocadas por cada par devem ser mantidos.
- Se um par de usuários não trocou nenhuma mensagem nos últimos k lotes de mensagens, sua lista de espera deve ser liberada no servidor para evitar o consumo desnecessário de recursos.
- Pode ocorrer de um lote conter as mensagens, entre um mesmo par, com identificador de ordem 3 e 4 e a mensagem com identificador 2 ainda não tiver sido enviada e nem estar neste lote. Neste caso, o servidor deve manter as mensagens 3 e 4 na lista de espera e esperar pela chegada da mensagem 2 nos lotes posteriores.

- Um par de usuários pode iniciar uma conversa a qualquer momento e não necessariamente no primeiro lote de mensagens.

2.1 Exemplo

Supondo que o simulador acabou de ser inciado, e receba o seguinte lote de mensagens.

```
3 2 Estou fazendo agora
0 1 Olá, tudo bem com você?
1 2 Não
1 1 Você vem jantar hoje?
0 2 Tudo sim e você?
1 3 Vou ter que trabalhar até mais tarde
2 4 se você lembrar
3 1 Conseguiu fazer o TP?
3 3 Esta muito fácil
```

Onde o primeiro inteiro identifica o par de usuários e o segundo a ordem da mensagem. Neste exemplo na mensagem:

```
3 2 Estou fazendo agora
```

O `3` indica que o par que está trocando esta mensagem é o par de identificador 3, o `2` indica que esta foi a segunda mensagem trocada por este par e `Estou fazendo agora` é o conteúdo da mensagem.

O servidor irá receber este lote, alocar 4 listas de espera de mensagens e distribuir as mensagens da seguinte forma:

```
Par 0: [(1,Olá, tudo bem com você?), (2,Tudo sim e você?)]
Par 1: [(1,Você vem jantar hoje?), (2,Não), (3,Vou ter que trabalhar até mais tarde)]
Par 2: [(4,se você lembrar)]
Par 3: [(1,Conseguiu fazer o TP?), (2,Estou fazendo agora), (3,Esta muito fácil)]
```

*Cada parentese representa uma mensagem. O primeiro elemento do parentese é o identificador de ordem da mensagem e o segundo o conteúdo.

Recebido todo o lote, ele deverá enviar as mensagens respeitando as restrições definidas na seção anterior. A principal restrição é que o par de usuários que trocou menos mensagens até agora, tem prioridade sobre os outros. Como nenhum par trocou nenhuma mensagem até agora todos possuem a mesma prioridade. O critério de desempate diz que em caso de empate, o par com menor identificador terá prioridade. Neste caso as mensagens serão enviadas na seguinte ordem:

Ordem	Número de mensagens trocadas	Mensagem escolhida
1	[0,0,0,0]	0 1 Olá, tudo bem com você?
2	[1,0,0,0]	1 1 Você vem jantar hoje?
3	[1,1,0,0]	3 1 Conseguiu fazer o TP?
4	[1,1,0,1]	0 2 Tudo sim e você?
5	[2,1,0,1]	1 2 Não
6	[2,2,0,1]	3 2 Estou fazendo agora
7	[2,2,0,2]	1 3 Vou ter que trabalhar até mais tarde
8	[2,3,0,2]	3 3 Esta muito fácil

*Note que a mensagem *"2 4 se você lembrar"* não foi enviada pois as mensagens 1, 2, e 3 da conversa deste par ainda não foi enviada. *O vetor na segunda coluna representa o número de mensagens que cada par já trocou entre si. A primeira posição é referente ao par com identificador 0, a segunda ao par com identificador 1 e assim sucessivamente.

Após este envio, as listas de espera deverão estar da seguinte forma:

```
Par 0: []
Par 1: []
Par 2: [(4,se você lembrar)]
Par 3: []
```

Como não a mais mensagens a ser enviada, o servidor espera um novo lote de mensagem chegar. Suponha que este lote seja:

```
2 2 Vou
2 3 leva na aula amanha
0 3 Tudo bem tbm
4 2 Parabéns! Que dia vamos comemorar?
2 1 Vai precisar da grana?
4 1 Passei no vestibular!
3 4 Esta mesmo
```

Note que temos um novo par de usuários se comunicando, o par 4. Assim uma nova lista de espera deve ser alocada. Após distribuir estas mensagens para suas respectivas listas, temos:

```
Par 0: [(3,Tudo bem tbm)]
Par 1: []
Par 2: [(1,Vai precisar da grana?),(2,Vou),(3,leva na aula amanha),(4,se você lembrar)]
Par 3: [(4,Esta mesmo)]
Par 4: [(1,Passei no vestibular!),(2,Parabéns! Que dia vamos comemorar?)]
```

Distribuídas todas as mensagens a ordem de envio será:

Ordem	**Número de mensagens trocadas**	**Mensagem escolhida**
1	[2,3,0,3,0]	2 1 Vai vai precisar da grana?
1	[2,3,1,3,0]	4 1 Passei no vestibular!
2	[2,3,1,3,1]	2 2 Vou
2	[2,3,2,3,1]	4 2 Parabéns! Que dia vamos comemorar?
3	[2,3,2,3,2]	0 3 Tudo bem tbm
4	[3,3,2,3,2]	2 3 leva na aula amanha
5	[3,3,3,3,2]	2 4 se você lembrar
6	[3,3,4,3,2]	3 4 Esta mesmo

*O vetor na segunda coluna representa o número de mensagens que cada par já trocou entre si. A primeira posição é referente ao par com identificador 0, a segunda ao par com identificador 1 e assim sucessivamente.
*Note que estes contadores de mensagens foram mantidos de um lote para o outro

E desta vez, após o envio, todas as listas de espera estarão vazias:

```
Par 0: []
Par 1: []
Par 2: []
Par 3: []
Par 4: []
```

Note que o par 1 não trocou nenhuma mensagem neste lote. Se isto voltar a acontecer em outros $k - 1$ lotes consecutivos sua lista de espera deverá ser desalocada. Exemplo considerando que os outros pares continuaram a trocar mensagens:

```
Par 0: [...]
Par 2: [...]
Par 3: [...]
Par 4: [...]
```

* As reticencias foram utilizadas para representar um conteúdo arbitrário nas listas.

3 Entrada e saída

A entrada será realizada através da entrada padrão *stdin*. Primeiramente será informados um inteiro $1 \leq k \leq 1000$ que indica o número máximo de lotes recebidos que um par de usuários pode ficar sem trocar mensagens. Exemplo: se $k = 7$, o servidor acabou de receber o 9º lote de mensagens e o par i só trocou mensagens no primeiro e segundo lote, a lista de espera deste par deverá ser desalocada.

Posteriormente serão passados os lotes de mensagens. Cada lote se inicia com a string `Lote` i onde $1 \leq i \leq 10000$ é o número do lote e termina com a string `Fim`. Dentro de cada lote serão passadas as mensagens. Cada mensagem será enviada em uma linha e contém três campos separados por ponto e virgula (;). O primeiro campo será um identificador inteiro do par de usuários que está trocando a mensagem, o segundo o identificador inteiro de ordem da mensagem e o terceiro o conteúdo: `ID_Par;ID_Ordem;Conteudo`. O tamanho máximo de conteúdo de uma mensagem é de 500 caracteres.

Para informar que não serão enviados mais lotes será passado uma linha contendo `-1`.

Exemplo de entrada com $k = 10$ e os dois lotes utilizados no exemplo anterior ficam da seguinte forma:

```
10
Lote 1
3;2;Estou fazendo agora
0;1;Olá, tudo bem com você?
1;2;Não
1;1;Você vem jantar hoje?
0;2;Tudo sim e você?
1;3;Vou ter que trabalhar até mais tarde
2;4;se você lembrar
3;1;Conseguiu fazer o TP?
3;3;Esta muito fácil
Fim
```

```
Lote 2
2 2 Vou
2 3 leva na aula amanha
0 3 Tudo bem tbm
4 2 Parabéns! Que dia vamos comemorar?
2 1 Vai precisar da grana?
4 1 Passei no vestibular!
3 4 Esta mesmo
Fim
-1
```

A saída utilizada será a saída padrão *stdout*. Como especificado, o servidor deve ler um lote completo de mensagens, armazenar estas mensagens nas listas apropriadas, definir a ordem de envio e enviar estas mensagens, após o envio de todas as mensagens possíveis ele deve ler o próximo lote e assim sucessivamente. Para este trabalho o seu simulador deve imprimir na saída padrão nas seguinte ocasiões:

- Ao término do recebimento do lote i imprima:

  `Lote_`i`:`
  `Listas:`

- Imprima o conteúdo de todas as listas de espera existentes. Para cada par i de usuários, o simulador irá imprimir a lista de espera associada a este par no seguinte formato:

  `Par_`i`:[(ID_Ordem,Conteudo),(ID_Ordem,Conteudo),...,(ID_Ordem,Conteudo)]`

  Onde i é o identificador do par e o conteúdo de cada parenteses representa uma mensagem na lista. Note que **não existe nenhum espaço** entre os campos.

  A ordem de impressão das listas deverá ser a ordem crescente em relação ao identificador do par associado a lista. Lembre-se que **as mensagens em uma mesma lista devem estar ordenadas crescentemente pelo identificador da ordem**. Note que as listas devem ser impressas antes do envio das mensagens.

- Terminado a impressão das listas, imprima:

  `Envios:`

- Ao selecionar uma mensagem a ser enviada, o servidor deve imprimir esta mensagem no seguinte formato:

  `ID_Par;ID_Ordem;Conteudo`

- Ao enviar todas as mensagens de um lote, deverá ser impresso:

  `Contadores:`

  E posteriormente imprima o contador de mensagens trocadas por cada par a cada momento no seguinte formato:

  `Par_`i`:`*count*

Onde i é o identificador do par e $count$ um valor inteiro que informa quantas mensagens foram trocadas pelo par i.

Novamente a ordem de impressão será a ordem crescente de acordo com identificador do par.

Note que estas informações deverão ser impressas para cada lote de mensagens. A saída esperada para a entrada de exemplo seria a seguinte:

```
Lote_1:
Listas:
Par_0:[(1,Olá, tudo bem com você?), (2,Tudo sim e você?)]
Par_1:[(1,Você vem jantar hoje?), (2,Não), (3,Vou ter que trabalhar até mais tarde)]
Par_2:[(4,se você lembrar)]
Par_3:[(1,Conseguiu fazer o TP?), (2,Estou fazendo agora), (3,Esta muito fácil)]
Envios:
0;1;Olá, tudo bem com você?
1;1;Você vem jantar hoje?
3;1;Conseguiu fazer o TP?
0;2;Tudo sim e você?
1;2;Não
3;2;Estou fazendo agora
1;3;Vou ter que trabalhar até mais tarde
3;3;Esta muito fácil
Contadores:
Par_0:2
Par_1:3
Par_2:0
Par_3:3
Lote_2:
Listas:
Par_0:[(3,Tudo bem tbm)]
Par_1:[]
Par_2:[(1,Vai precisar da grana?),(2,Vou),(3,leva na aula amanha),(4,se você lembrar)]
Par_3:[(4,Esta mesmo)]
Par_4:[(1,Passei no vestibular!),(2,Parabéns! Que dia vamos comemorar?)]
Envios:
2;1;Vai precisar da grana
4;1;Passei no vestibular!
2;2;Vou
4;2;Parabéns! Que dia vamos comemorar?
0;3;Tudo bem tbm
2;3;leva na aula amanha
2;4;se você lembrar
3;4;Esta mesmo
Contadores:
Par_0:3
Par_1:3
Par_2:4
Par_3:4
Par_4:2
```

Observações:

- **Não imprima espaços em branco ao final das linhas**;
- Os únicos espaços em branco que devem existir são os espaços dentro do conteúdo da mensagem.

4 Detalhes de Implementação

A boa prática de programação indica que o código desenvolvido deve ser legível, bem modularizado e comentado.

- Legível: Nomes intuitivos para funções e variáveis, identação, quebras de linhas adequadas, etc.
- Modularização: O código deve ser decomposto em funções e estruturas semanticamente coesas.
- Comentários: Trechos relevantes e/ou complexos do código devem ser comentados para facilitar a compreensão do código.

Todas as estruturas utilizadas devem ser alocadas dinamicamente. Todas as estruturas alocadas que não forem mais necessárias, devem ser liberadas (utilizando *free*).

Uma boa ferramenta para avaliar se a alocação e liberação de memória foi realizada corretamente é o *valgrind*[3].

5 O que deve ser entregue

Você deve submeter uma documentação de até 10 páginas contendo uma descrição da solução proposta, detalhes relevantes da implementação, além de uma análise de complexidade de tempo dos algoritmos envolvidos e de complexidade de espaço das estruturas de dados utilizadas.

A documentação deve conter:

1. Introdução: descrição sucinta do problema a ser resolvido e visão geral sobre o funcionamento do programa.
2. Implementação: descrição sobre a implementação do programa. Devem ser detalhados as estruturas de dados utilizadas (de preferência com diagramas ilustrativos) e o funcionamento das principais funções.
3. Estudo de complexidade: estudo da complexidade do tempo de execução dos procedimentos implementados.

[3] http://cs.ecs.baylor.edu/~donahoo/tools/valgrind/

4. Avaliação experimental do desempenho da sua implementação.

5. Conclusão: comentários gerais sobre o trabalho e as principais dificuldades encontradas em sua implementação.

6. Bibliografia: bibliografia utilizada para o desenvolvimento do trabalho, incluindo sites da Internet, se for o caso.

Utilizando o sistema VPL do MOODLE, você deverá submeter todos os seus códigos fonte (todos os arquivos .c e .h) do simulador. Também deverá ser entregue arquivo **.pdf** contendo a documentação (máximo 10 páginas).

6 Comentários gerais

- Comece a fazer este trabalho logo, enquanto o problema está fresco na memória e o prazo para terminá-lo está tão longe quanto jamais poderá estar.
- Clareza, identação e comentários no programa também vão valer pontos.
- Siga atentamente os formatos de entrada e saída especificados. **Não imprima espaços em branco ao final das linhas**.
- Essa especificação não é isenta de erros e ambiguidades. Portanto, se tiverem problemas para entender o que está escrito aqui, pergunte aos monitores.
- Você **não** precisa de utilizar uma IDE como Netbeans, Eclipse, Code::Blocks ou QtCreator para implementar esse trabalho prático. No entanto, se o fizer, **não** inclua os arquivos extras que foram gerados por essa IDE em sua submissão.
- Trabalhos copiados serão penalizados com a nota zero. Serão utilizadas ferramentas automáticas de detecção de cópias.
- Penalização por atraso: (2d - 1) pontos, em que d é o número de dias de atraso.