Gestão da conexão

Grupo de Sistemas Distribuídos
Universidade do Minho

Objetivos

Gestão de uma conexão para suporte de diferentes padrões de interação por processos multi-threaded (clientes e servidores): delimitação de mensagens nos streams; etiquetamento de mensagens para, por exemplo, distinguir tipos de interacção ou permitir associação entre pedidos e respostas; demultiplexing de mensagens etiquetadas.

Exercícios propostos

1 Observe os códigos de exemplo de `SimpleClient` e `SimpleServerWithWorkers`. Nestes, o servidor tem várias threads concorrentes a ler mensagens de cada cliente e enviar respostas. Para tal faz uso de uma classe `FramedConnection`:

```
public class FramedConnection implements AutoCloseable {
    public FramedConnection(Socket socket) throws IOException { ... }
    public void send(byte[] data) throws IOException { ... }
    public byte[] receive() throws IOException { ... }
    public void close() throws IOException { ... }
}
```

Implemente `FramedConnection`. Esta deve permitir cada uma das partes (cliente e servidor) trocarem mensagens entre si. A classe é agnóstica ao conteúdo das mensagens, para poderem ser usados diferentes formatos de serialização. Deve ser permitido que várias threads invoquem os métodos concorrentemente. De notar que os dois sentidos de comunicação são independentes, devendo ser possível uma thread estar a ler e outra a escrever simultanemente.

Sugestão: para delimitar as mensagens, envie o tamanho do conteúdo (do byte[]) seguido deste. Utilize `Data[Input|Output]Streams` e os métodos de ler/escrever inteiros e arrays de bytes. Em particular, utilize `readFully()`.

2 No exemplo anterior não há qualquer associação entre pedidos e respostas, chegando estas por ordem arbitrária. No sentido de permitir tal associação, é útil atribuir uma etiqueta a cada mensagem. Esta pode representar, por exemplo, tipos de processamento desejado ou um identificador de thread cliente. A classe `TaggedConnection` especializa a `FramedConnection`, sendo agora o conteúdo de cada *frame* uma etiqueta (inteiro) seguida de dados.

```
public class TaggedConnection implements AutoCloseable {
    public static class Frame {
        public final int tag;
        public final byte[] data;
        public Frame(int tag, byte[] data) { this.tag = tag; this.data = data; }
    }
    public TaggedConnection(Socket socket) throws IOException { ... }
    public void send(Frame frame) throws IOException { ... }
    public void send(int tag, byte[] data) throws IOException { ... }
    public Frame receive() throws IOException { ... }
    public void close() throws IOException {
}
```

Escreva a classe `TaggedConnection`, e teste com os exemplos de código `SequentialClient` e `ServerWithWorkers`. Nestes exemplos é convencionado que uma mensagem com etiqueta 0 não é respondida, para as etiquetas impares é devolvida uma resposta (com a mesma etiqueta) e para as pares é enviada para o cliente uma sequência de mensagens, terminada por uma de conteúdo vazio. Repare que isto pode funcionar como pretendido dado o cliente ser sequencial.

3 Corra o exemplo `WrongThreadedClient` com o `ServerWithWorkers`. Como agora o cliente é multi-threaded, o comportamento pretendido não se vai verificar, não havendo garantia que as respostas cheguem às threads apropriadas. Ao correr com asserções ligadas (`java -ea`) estas vão falhar.

Para permitir que as mensagens que chegam sejam encaminhadas para as threads appropriadas, é introduzida uma nova abstracção, o `Demultiplexer`, usado no exemplo `ThreadedClient`:

```
public class Demultiplexer implements AutoCloseable {
   public Demultiplexer(TaggedConnection conn) { ... }
   public void start() { ... }
   public void send(Frame frame) throws IOException { ... }
   public void send(int tag, byte[] data) throws IOException { ... }
   public byte[] receive(int tag) throws IOException, InterruptedException { ... }
   public void close() throws IOException { ... }
}
```

Esta classe encapsula uma `TaggedConnection` e, para além de delegar o envio de mensagens para esta, disponibiliza a operação `receive(int tag)`, que deve bloquear a thread invocadora até chegar uma mensagem com a etiqueta especificada, devolvendo então o conteúdo. Teste com o `ThreadedClient` e `ServerWithWorkers`.

Sugestões: Crie uma thread responsável por ler do `TaggedConnection` aquando do `start()`. Para permitir threads ficarem bloqueadas à espera de uma dada etiqueta, utilize um mapa de inteiros para objectos contendo uma variável de condição, e onde serão depositadas as mensagens com essa etiqueta, numa `Dequeue`.

Exercícios adicionais

4 Reimplemente `Demultiplexer`, mas sem criar thread para ler do `TaggedConnection` (e sem método `start()`), sendo as leituras feitas pelas próprias threads que usam o `Demultiplexer`.