Guião N.º 3

Servidor TCP com múltiplos clientes (em Java)

Sistemas Distribuídos

Ricardo Costa
rcosta@estg.ipp.pt

ESTGF | POLITÉCNICO DO PORTO

Outubro 2017

1 Suporte múltiplos clientes (em simultâneo)

Um *socket* TCP suporta naturalmente vários clientes, contudo, e assumindo o exemplo simplista no Guião N.º1, o processamento dos pedidos dos clientes é efectuado de forma sequencial. Ou seja, o primeiro cliente a estabelecer a ligação poderá interagir com o servidor, os restantes clientes que entretanto cheguem serão colocados em fila de espera, até que o cliente actual termine a sua ligação. Os clientes são então processados por ordem de chegada, mas apenas um de cada vez.

Este modelo de operação tem várias desvantagens, nomeadamente o atraso que gera no atendimento dos clientes. Tal atrasado pode inclusive levar a que os clientes terminem a sua tentativa de estabelecimento de ligação por *timeout*. Nestas situações é importante aumentar a eficiência do servidor, recorrendo por exemplo ao *threads*. Tal permitirá o atendimento simultâneo de múltiplos clientes, aumentando a eficiência global do servidor.

1.1 *Threads*

Uma *thread* retrata uma linha de execução de um processo. Um processo têm no mínimo uma linha de execução, podendo no entanto ter mais do que

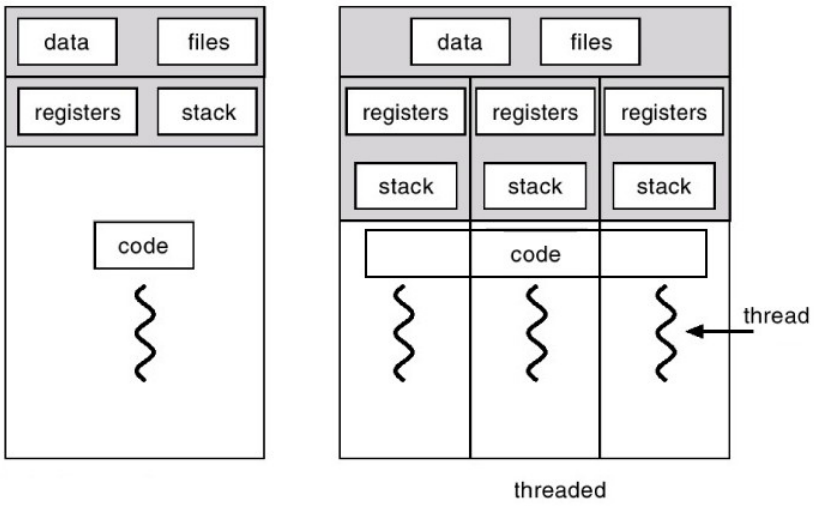

Figura 1: Comparação entre processos e *threads* (em *Unix Threads*)

uma. A Figura 1 mostra a relação entre processos e *threads*. Na figura pode se ver que todas as *threads* de um processo, partilham o contexto do mesmo processo (i.e. PID, espaço de memória, variáveis globais, código, ...).

Então, uma *thread* não é mais do que uma nova execução no contexto de um mesmo processo. Uma *thread* pode, no entanto, executar código diferente das restantes *threads* desde que esse código seja parte do código do processo original.

1.2 *Threads* em Java

Uma forma simples de se implementar *threads* em Java é estendendo a classe *Thread*. Desta forma consegue-se concentrar a codificação do trabalho a efectuar pelo servidor, a quando do estabelecimento de uma ligação com um cliente, numa nova classe.

```
1  import java.io.*;
2  import java.net.*;
3
4  public class WorkerThread extends Thread{
5      private Socket socket = null;
6
7      public WorkerThread(Socket socket) {
8    super("WorkerThread");
9    this.socket = socket;
10     }
11
```

```
12      public void run() {
13    try {
14        PrintWriter out = new PrintWriter(socket.getOutputStream()
              , true);
15        BufferedReader in = new BufferedReader(
16              new InputStreamReader(
17              socket.getInputStream()));
18        String inputLine;
19
20        while ((inputLine = in.readLine()) != null) {
21      out.println(inputLine);
22      if (inputLine.equals("Bye"))
23          break;
24        }
25        out.close();
26        in.close();
27        socket.close();
28
29    } catch (IOException e) {
30        e.printStackTrace();
31    }
32      }
33  }
```

Listing 1: Exemplo de uma classe que implementa *threads*

A Listagem 1 apresenta um exemplo de uma classe que estende a classe *Thread* (linha 4). É de notar em particular o construtor desta classe (linhas 7 a 10), já que a execução do método *super()* é o que permite à classe disponibilizar-se para execução. Já o método *run()* (linha 12) é o análogo ao método *main()* de um processo. O método *run()* é então executado sempre que se invoca o método *start()* (ver linha 15 da Listagem 2) de um objecto de uma classe *Thread* (ou derivada desta).

```
1  public class TcpMultiServer {
2      public static void main(String[] args) throws IOException {
3          ServerSocket serverSocket = null;
4          int port=2048;
5          boolean listening = true;
6
7          try {
8              serverSocket = new ServerSocket(port);
9          } catch (IOException e) {
10             System.err.println("Could not listen on port: "+port
                   +".");
11             System.exit(-1);
12         }
13
```

```
14            while (listening)
15          new WorkerThread(serverSocket.accept()).start();
16
17            serverSocket.close();
18        }
19    }
```

Listing 2: Exemplo de servidor TCP (multi-*thread*)

2 Exercícios

1. Implemente o código de um Servidor Echo com o suporte para múltiplos clientes simultâneos.
   Teste o seu servidor usando o comando *telnet* várias vezes, em simultâneo.

2. Desenvolva um par de aplicações cliente e servidor para *chat* em rede, em que:

Servidor Ao servidor compete a recepção e envio das mensagens para todos os clientes. Antes do envio das mensagens, o servidor deve lhes adicionar a data e hora, bem como o IP do cliente que enviou a mensagem.

Cliente Ao cliente compete a impressão de todas as mensagens recebidas do servidor, bem como o envio de mensagens digitadas pelo seu utilizador. Sugere-se que seja implementada uma *thread* para a recepção e impressão de mensagens, e outra *thread* para a obtenção de mensagens do utilizador e respectivo envio para o servidor.

3 Recursos *online*

- The Java Tutorial, Networking
  `http://download.oracle.com/javase/tutorial/networking/`
- The Java Tutorial, Networking, All About Sockets
  `http://download.oracle.com/javase/tutorial/networking/sockets/`
- Unix Threads
  `http://www.cs.miami.edu/~geoff/Courses/CSC322-11S/Content/UNIXProgramming/UNIXThreads.shtml`