

# Sistemas Distribuídos - Relatório 1

Claudio Quessada Cabello

4 de novembro de 2020

# Sumário

<b>Lista de Figuras</b>	<b>2</b>
<b>Aula prática 1</b>	<b>3</b>
1. Executar os servidores . . . . .	3
1.1 Usar VisualVM . . . . .	4
1.2 Usar Wireshark . . . . .	5
2. Adicionar o pool de threads . . . . .	6

# Lista de Figuras

1	Execução do servidor baseado em threads . . . . .	3
2	Execução do servidor baseado em NIO . . . . .	3
3	Execução do servidor baseado em fork . . . . .	3
4	Threads do servidor baseado em threads . . . . .	4
5	Threads do servidor baseado em NIO . . . . .	4
6	Captura de pacotes do servidor baseado em threads . . . . .	5
7	Captura de pacotes do servidor baseado em NIO . . . . .	5
8	Captura de pacotes do servidor baseado em fork . . . . .	5
9	Estrutura do servidor baseado em threadpool . . . . .	6
10	Implementação dos métodos TCPThread.work() e main() . . . . .	7
11	Threads do servidor baseado em threadpool . . . . .	7
12	Captura de pacotes do servidor baseado em threadpool . . . . .	8

# Aula prática 1

## 1. Executar os diferentes servidores em uma máquina diferente da máquina cliente.

Os servidores foram executados em duas máquinas Windows conectadas via Ethernet. O servidor em C foi executado em um subsistema Linux do windows (WSL). Podemos notar pelas figuras abaixo que o desempenho do código utilizando NIO foi muito superior aos outros dois servidores. Por questões de compatibilidade com a IDE e problemas com o uso de pacotes, as classes em java foram renomeadas, mas são, à parte disso, as mesmas do enunciado.

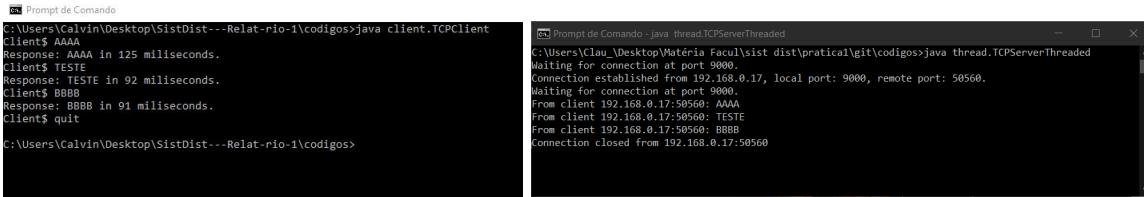


Figura 1: Execução do servidor baseado em threads

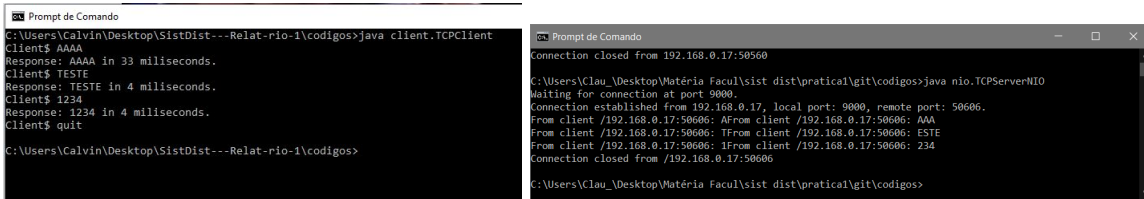


Figura 2: Execução do servidor baseado em NIO

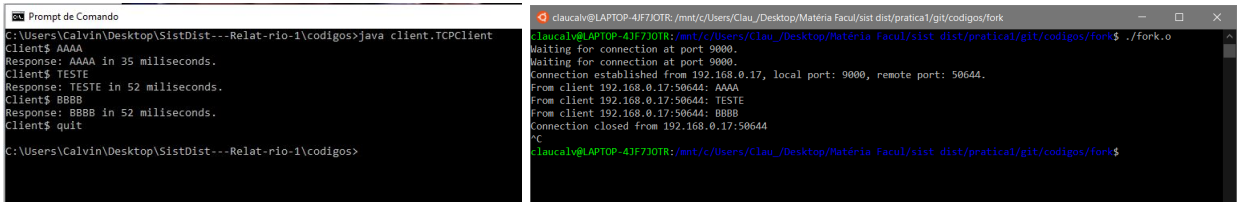


Figura 3: Execução do servidor baseado em fork

1.1 - Usar o Visual VM para avaliar o comportamento dos programas em Java, incluindo o desempenho.

Além da resposta mais rápida, percebe-se que o servidor NIO não possui o custo das threads extras que o threaded possui.

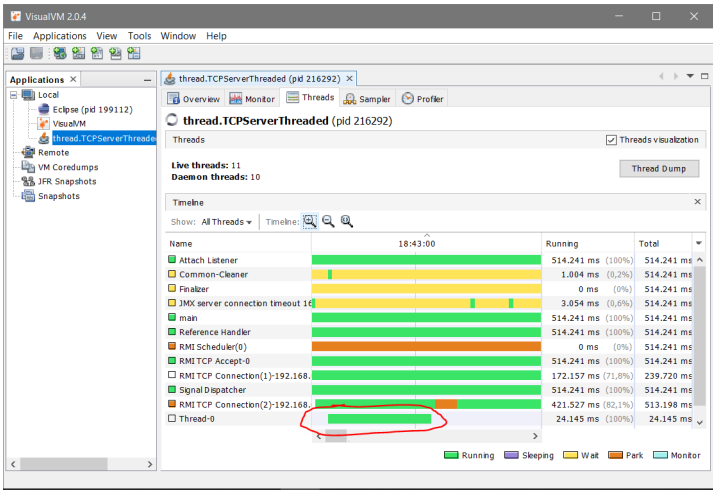


Figura 4: Threads do servidor baseado em threads

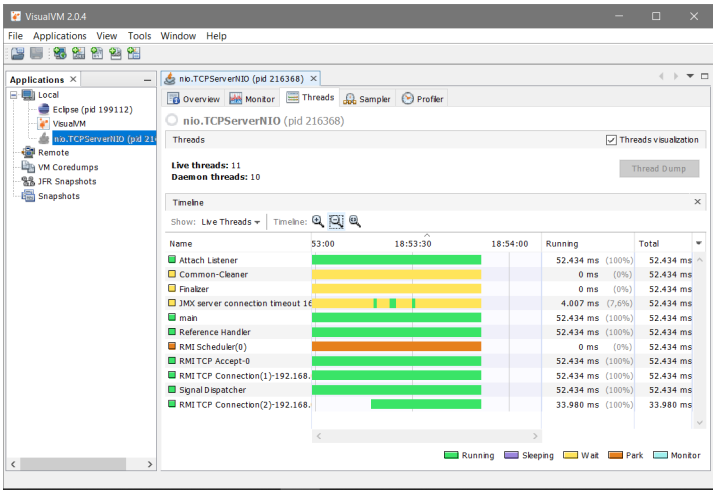


Figura 5: Threads do servidor baseado em NIO

1.2 - Fazer a captura dos pacotes usando Wireshark e discutir o protocolo de aplicação usado.

Podemos ver todos os sinais de um protocolo TCP, a começar pelo handshake de 3 vias, os pacotes de confirmação entre os pacotes de mensagem, e a finalização bidirecional da comunicação.

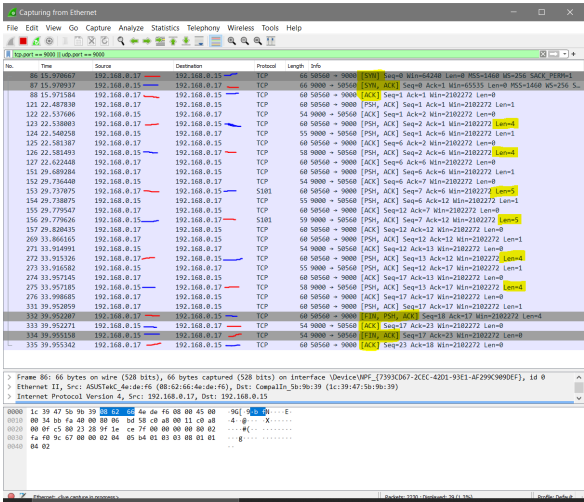


Figura 6: Captura de pacotes do servidor baseado em threads

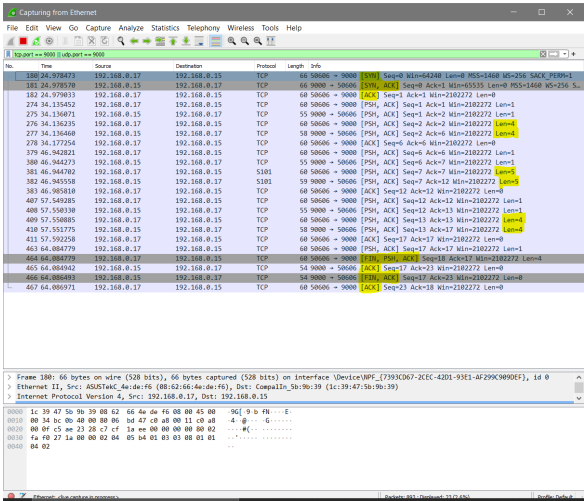


Figura 7: Captura de pacotes do servidor baseado em NIO

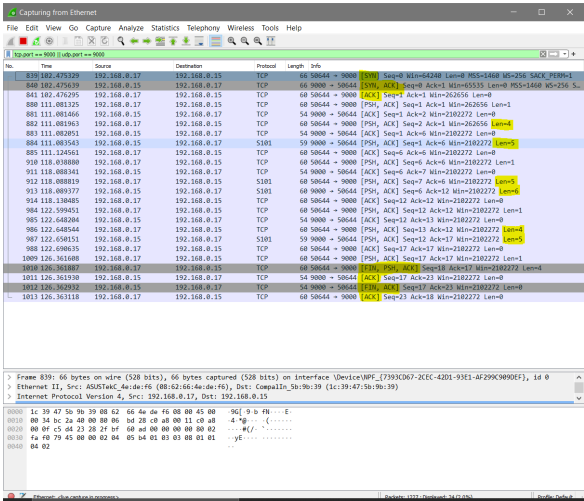


Figura 8: Captura de pacotes do servidor baseado em fork

## 2. Modificar o servidor Java multi-threaded para usar pool de threads.

A estrutura base foi adaptada do tutorial sugerido pelo professor , e as principais implementações, na outra figura abaixo, retiradas do código original TCPServerThreaded.

```
1 public class TCPServerThreadPool{
2     /** Well-known server port. */
3     public static int serverPort = 9000;
4
5     private class TCPThread extends Thread { // Deveria ter colocado em outro arquivo? Provavelmente sim.
6
7         private boolean isStopped = false;
8
9         public void run() {
10             while(!isStopped()){
11                 try{
12                     work(taskQueue.take());
13                 } catch(Exception e){
14                     //log or otherwise report exception,
15                     //but keep pool thread alive.
16                 }
17             }
18         }
19
20         private void work(Socket server){
21         }
22
23         public synchronized void doStop(){
24             isStopped = true;
25             this.interrupt(); //break pool thread out of dequeue() call.
26         }
27
28         public synchronized boolean isStopped(){ return isStopped; }
29     }
30
31     private final BlockingQueue<Socket> taskQueue;
32     private final List<TCPThread> threads;
33     private boolean isStopped = false;
34
35     public TCPServerThreadPool(int noOfThreads, int maxNoOfTasks){
36
37         taskQueue = new ArrayBlockingQueue<>(maxNoOfTasks);
38         threads = new ArrayList<TCPThread>();
39
40         for(int i=0; i<noOfThreads; i++){
41             threads.add(new TCPThread());
42         }
43
44         for(TCPThread thread : threads)
45             thread.start();
46     }
47
48     public synchronized void work(Socket socket) throws Exception{
49         if(isStopped) throw
50             new IllegalStateException("ThreadPool is stopped");
51
52         taskQueue.put(socket);
53     }
54
55     public synchronized void stop(){
56         isStopped = true;
57         for(TCPThread thread : threads)
58             thread.doStop();
59     }
60 }
```

Figura 9: Estrutura do servidor baseado em threadpool

```

1 private void work(Socket server){
2     try {
3         // Create a BufferedReader object to read strings from the socket. (read strings FROM CLIENT)
4         BufferedReader br = new BufferedReader(new InputStreamReader(server.getInputStream()));
5         String input = br.readLine();
6         //Create output stream to write to/send TO CLIENT
7         DataOutputStream output = new DataOutputStream(server.getOutputStream());
8         //Keep repeating until the command "quit" is read.
9         while (!input.equals("quit")) {
10            //Convert input to upper case and echo back to client.
11            System.out.println("From client " + server.getInetAddress().getHostAddress() + ":" + server.getPort() + ": " + input);
12            output.writeBytes(input.toUpperCase() + "\n");
13            input = br.readLine();
14        }
15        System.out.println("Connection closed from " + server.getInetAddress().getHostAddress() + ":" + server.getPort());
16        //Close current connection
17        br.close();
18        output.close();
19        server.close();
20    } catch (IOException e) {
21        //Print exception info
22        e.printStackTrace();
23    }
24 }

25
26 @SuppressWarnings("resource")
27 public static void main (String args[]) throws Exception {
28
29     TCPServerThreadPool ts = new TCPServerThreadPool(10, 20);
30     //Dispatcher socket
31     ServerSocket serverSocket = new ServerSocket(serverPort);
32     //Waits for a new connection. Accepts connection from multiple clients
33     while (true) {
34         System.out.println("Waiting for connection at port " + serverPort + ".");
35         //Worker socket
36         Socket s = serverSocket.accept();
37         System.out.println("Connection established from " + s.getInetAddress().getHostAddress() + ", local port: " + s.getLocalPort() + ", remote port: " + s.getPort() + ".");
38         //Invoke the worker thread
39         ts.work(s);
40     }
41
42     //ts.stop(); Unreachable por causa do while(true)
43 }
44 }

```

Figura 10: Implementação dos métodos TCPThread.work() e main()

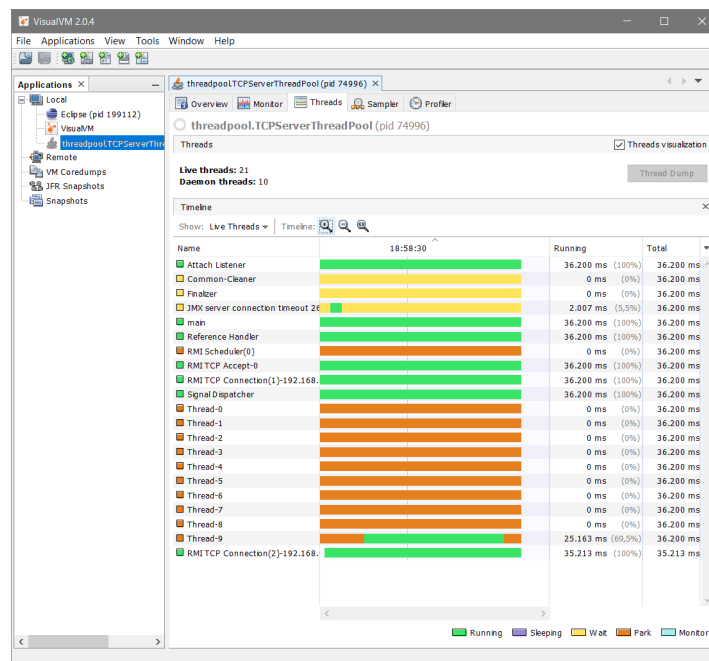


Figura 11: Threads do servidor baseado em threadpool



Capturing from Ethernet									
No.	Time	Source	Destination	Protocol	Length	Info			
277	0.132751	192.168.0.17	192.168.0.15	TCP	66	58029 → 58029 [ACK] Seq=814248 Len=0 MSS=1460 Win=256 Sack=1780-1			
278	43.526896	192.168.0.15	192.168.0.17	TCP	66	58029 → 58029 [ACK] Seq=814248 Len=0 MSS=1460 Win=256 Sack=1780-1			
279	43.527757	192.168.0.17	192.168.0.15	TCP	66	58029 → 58029 [ACK] Seq=814248 Len=0 MSS=1460 Win=256 Sack=1780-1			
300	49.395118	192.168.0.17	192.168.0.15	TCP	68	58029 → 58029 [PSH, ACK] Seq=1 Min=2182272 Len=1			
301	49.447786	192.168.0.15	192.168.0.17	TCP	54	58029 → 58029 [ACK] Seq=1 Min=2182272 Len=0			
302	49.448365	192.168.0.17	192.168.0.15	TCP	68	58029 → 58029 [PSH, ACK] Seq=1 Min=2182272 Len=1			
303	49.458344	192.168.0.15	192.168.0.17	TCP	55	58029 → 58029 [PSH, ACK] Seq=1 Min=2182272 Len=0			
304	49.499965	192.168.0.17	192.168.0.15	TCP	68	58029 → 58029 [ACK] Seq=1 Min=2182272 Len=0			
305	49.491866	192.168.0.15	192.168.0.17	TCP	58	58029 → 58029 [PSH, ACK] Seq=1 Min=2182272 Len=1			
306	49.532143	192.168.0.17	192.168.0.15	TCP	68	58029 → 58029 [ACK] Seq=1 Min=2182272 Len=0			
401	59.868402	192.168.0.17	192.168.0.15	TCP	68	58029 → 58029 [PSH, ACK] Seq=1 Min=2182272 Len=1			
402	59.138829	192.168.0.15	192.168.0.17	TCP	54	58029 → 58029 [ACK] Seq=1 Min=2182272 Len=0			
403	59.139775	192.168.0.17	192.168.0.15	TCP	5181	58029 → 58029 [PSH, ACK] Seq=1 Min=2182272 Len=1			
404	59.140758	192.168.0.15	192.168.0.17	TCP	55	58029 → 58029 [PSH, ACK] Seq=1 Min=2182272 Len=0			
405	59.181936	192.168.0.17	192.168.0.15	TCP	68	58029 → 58029 [ACK] Seq=1 Min=2182272 Len=0			
406	59.182823	192.168.0.15	192.168.0.17	TCP	59	58029 → 58029 [PSH, ACK] Seq=1 Min=2182272 Len=1			
407	59.223146	192.168.0.17	192.168.0.15	TCP	68	58029 → 58029 [ACK] Seq=1 Min=2182272 Len=0			
427	64.851642	192.168.0.17	192.168.0.15	TCP	68	58029 → 58029 [PSH, ACK] Seq=1 Min=2182272 Len=1			
429	64.866207	192.168.0.15	192.168.0.17	TCP	54	58029 → 58029 [ACK] Seq=1 Min=2182272 Len=0			
430	64.866983	192.168.0.17	192.168.0.15	TCP	68	58029 → 58029 [PSH, ACK] Seq=1 Min=2182272 Len=1			
431	64.862618	192.168.0.15	192.168.0.17	TCP	55	58029 → 58029 [PSH, ACK] Seq=1 Min=2182272 Len=0			
432	64.184228	192.168.0.17	192.168.0.15	TCP	68	58029 → 58029 [ACK] Seq=1 Min=2182272 Len=0			
433	64.184268	192.168.0.15	192.168.0.17	TCP	58	58029 → 58029 [PSH, ACK] Seq=1 Min=2182272 Len=1			
435	64.145597	192.168.0.17	192.168.0.15	TCP	68	58029 → 58029 [ACK] Seq=1 Min=2182272 Len=0			
439	68.902556	192.168.0.17	192.168.0.15	TCP	68	58029 → 58029 [PSH, ACK] Seq=1 Min=2182272 Len=1			
440	68.903858	192.168.0.15	192.168.0.17	TCP	68	58029 → 58029 [ACK] Seq=1 Min=2182272 Len=0			
441	68.902996	192.168.0.15	192.168.0.17	TCP	54	58029 → 58029 [ACK] Seq=1 Min=2182272 Len=0			
442	68.903868	192.168.0.15	192.168.0.17	TCP	54	58029 → 58029 [ACK] Seq=1 Min=2182272 Len=0			
443	68.905411	192.168.0.17	192.168.0.15	TCP	68	58029 → 58029 [ACK] Seq=1 Min=2182272 Len=0			

Frame 277: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface \Device\NPF\_{7393C067-2CEC-4201-93E1-AF299C980DEF}, Id 0

Ethernet II, Src: ASUSTakAc:de:f6:08:62:66:4a:de:f6, Dst: CompaqIn:50:90:39:1c:39:47:50:90:39

Internet Protocol Version 4, Src: 192.168.0.17, Dst: 192.168.0.15

```

0000  1c 39 47 50 90 39 08 62 66 4a de f6 08 00 45 00  0C 0 b f6 ..E
0010  00 34 bc 19 48 00 00 00 bd 39 c8 a8 00 11 c0 a8  -4 0... 9.....
0020  00 6f c5 c5 23 28 7f c1 63 b2 00 00 00 00 00 00  -K c.....
0030  fa f0 26 1d 00 00 02 04 05 b4 01 03 08 01 01    -B.....
0040  04 02

```

Figura 12: Captura de pacotes do servidor baseado em threadpool