Sistemas Distribuídos - Relatório 1

Claudio Quessada Cabello

4 de novembro de 2020

Sumário

Lista de Figuras

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.

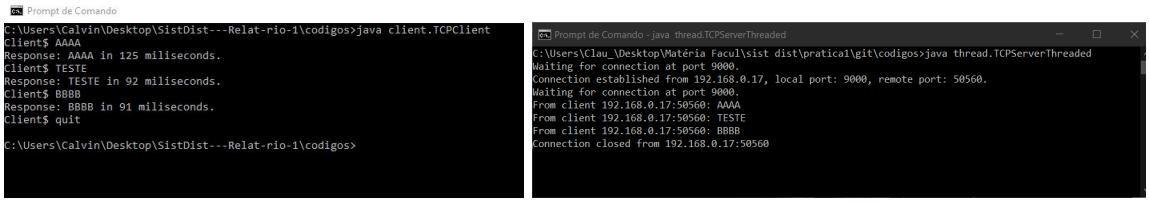

Figura 1: Execução do servidor baseado em threads

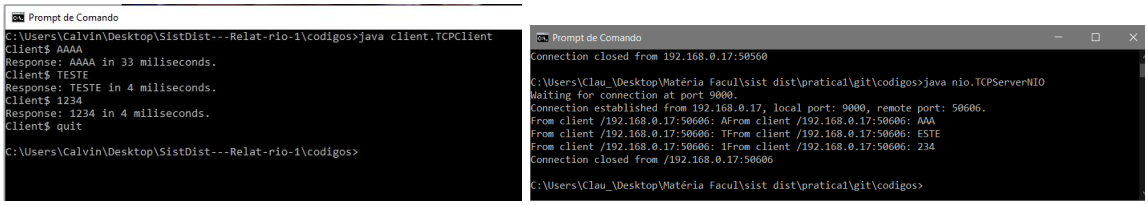

Figura 2: Execução do servidor baseado em NIO

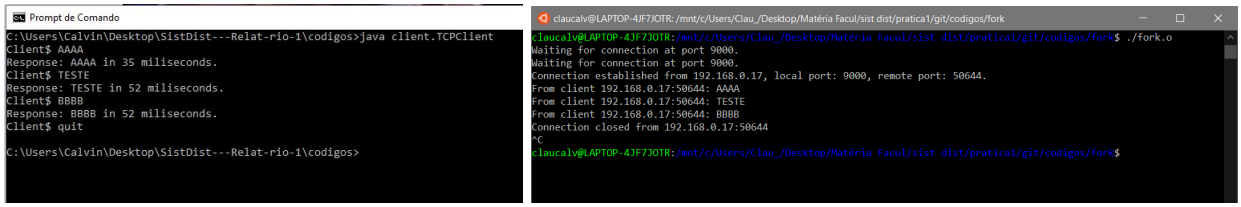

Figura 3: Execução do servidor baseado em fork

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.

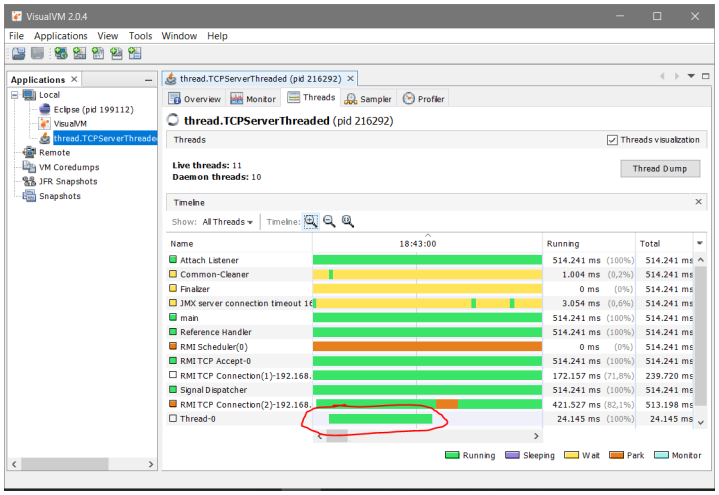

Figura 4: Threads do servidor baseado em threads

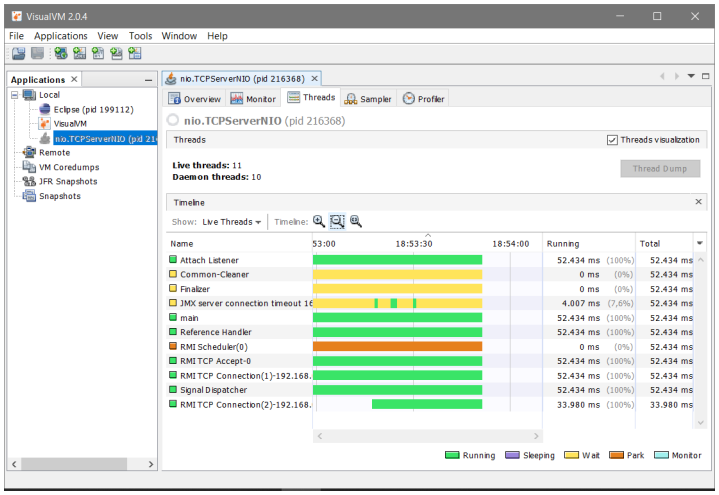

Figura 5: Threads do servidor baseado em NIO

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.

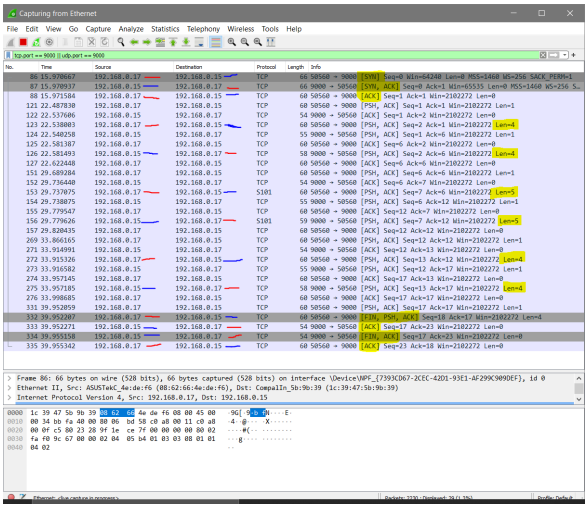

Figura 6: Captura de pacotes do servidor baseado em threads

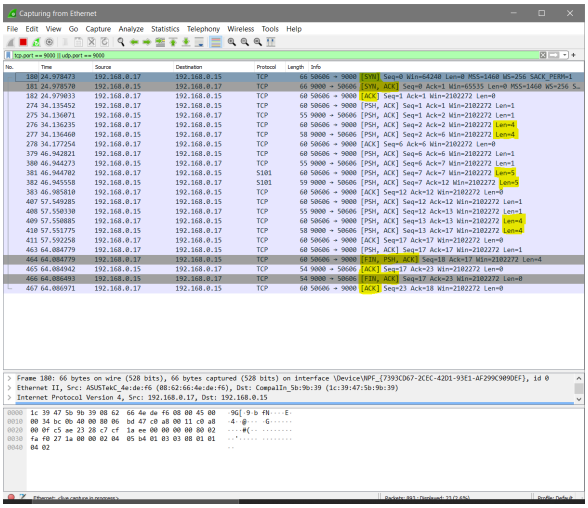

Figura 7: Captura de pacotes do servidor baseado em NIO

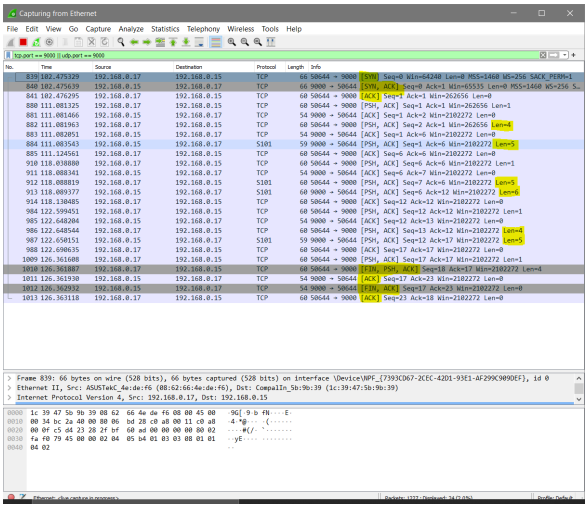

Figura 8: Captura de pacotes do servidor baseado em fork

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.

```
public class TCPServerThreadPool{
  /** Well-known server port. */
  public static int serverPort = 9000;

  private class TCPThread extends Thread { // Deveria ter colocado em outro arquivo? Provavelmente sim.

    private boolean isStopped = false;

    public void run() {
      while(!isStopped()){
        try{
          work(taskQueue.take());
        } catch(Exception e){
        //log or otherwise report exception,
        //but keep pool thread alive.
        }
      }
    }

    private void work(Socket server){
    }

    public synchronized void doStop(){
        isStopped = true;
        this.interrupt(); //break pool thread out of dequeue() call.
    }

    public synchronized boolean isStopped(){ return isStopped; }

  }

  private final BlockingQueue<Socket> taskQueue;
  private final List<TCPThread> threads;
  private boolean isStopped = false;

  public TCPServerThreadPool(int noOfThreads, int maxNoOfTasks){

    taskQueue = new ArrayBlockingQueue<>(maxNoOfTasks);
    threads = new ArrayList<TCPThread>();

    for(int i=0; i<noOfThreads; i++)
        threads.add(new TCPThread());

    for(TCPThread thread : threads)
        thread.start();

  }

  public synchronized void work(Socket socket) throws Exception{
    if(isStopped) throw
      new IllegalStateException("ThreadPool is stopped");

    taskQueue.put(socket);
  }

  public synchronized void stop(){
    isStopped = true;
    for(TCPThread thread : threads)
      thread.doStop();
  }
```

Figura 9: Estrutura do servidor baseado em threadpool

```java
    private void work(Socket server){
      try {
        // Create a BufferedReader object to read strings from the socket. (read strings FROM CLIENT)
        BufferedReader br = new BufferedReader(new InputStreamReader(server.getInputStream()));
        String input = br.readLine();
        //Create output stream to write to/send TO CLIENT
        DataOutputStream output = new DataOutputStream(server.getOutputStream());
        //Keep repeating until the command "quit" is read.
        while (!input.equals("quit")) {
          //Convert input to upper case and echo back to client.
          System.out.println("From client "+ server.getInetAddress().getHostAddress() +":"+server.getPort()+": " +input);
          output.writeBytes(input.toUpperCase() + "\n");
          input = br.readLine();
        }
        System.out.println("Connection closed from "+ server.getInetAddress().getHostAddress()+":"+server.getPort());
        //Close current connection
        br.close();
        output.close();
        server.close();
      } catch (IOException e) {
        //Print exception info
        e.printStackTrace();
      }
    }

  @SuppressWarnings("resource")
  public static void main (String args[]) throws Exception {

    TCPServerThreadPool ts = new TCPServerThreadPool(10, 20);
        //Dispatcher socket
    ServerSocket serverSocket = new ServerSocket(serverPort);
    //Waits for a new connection. Accepts connection from multiple clients
    while (true) {
      System.out.println("Waiting for connection at port "+serverPort+".");
      //Worker socket
      Socket s = serverSocket.accept();
      System.out.println("Connection established from " + s.getInetAddress().getHostAddress() + ", local port: "+s.getLocalPort
     ()+", remote port: "+s.getPort()+".");
      //Invoke the worker thread
      ts.work(s);
    }

    //ts.stop(); Unreachable por causa do while(true)
  }
}
```

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

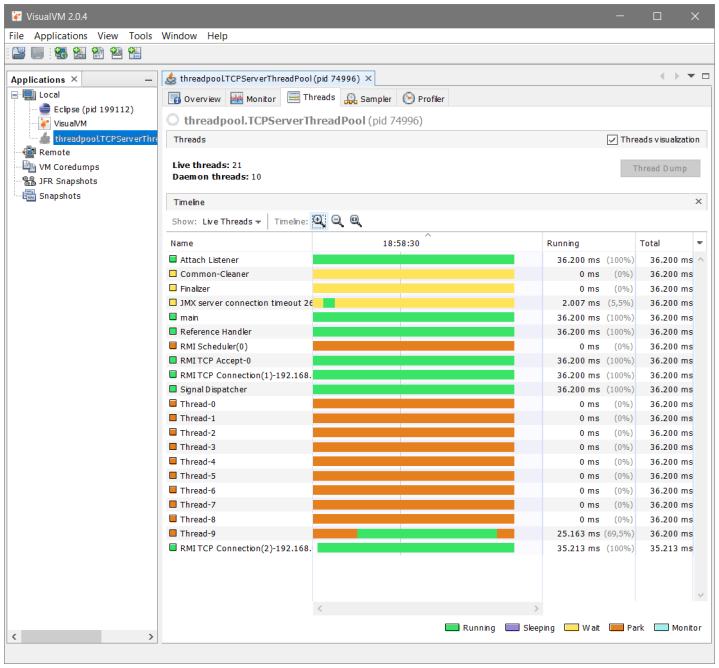

Figura 11: Threads do servidor baseado em threadpool

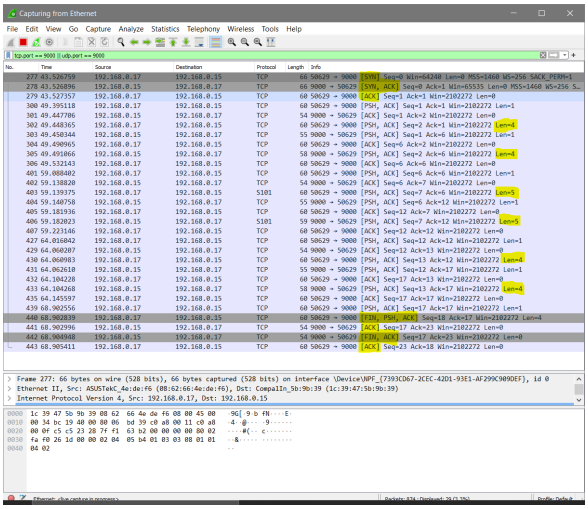

Figura 12: Captura de pacotes do servidor baseado em threadpool