Simulação de Pool de Impressão Dístribuida Utilizando Socket Exercício Computacional II - Sistemas Distribuídos

Rafael Gonçalves de Oliveira Viana[1]

[1]Sistemas de Informação – Universidade Federal do Mato Grosso do Sul (UFMS)
Caixa Postal 79400-000 – Coxim – MS – Brazil

rafael.viana@aluno.ufms.br

24 de outubro de 2017

Resumo. *Este documento relata os pontos fortes no desenvolvimento de uma Pool de Thread na linguaguem Java versão 8.*

1. Introdução

De acordo com [1], em um ambiente de trabalho, onde desejasse imprimir documentos enviados, a um curto período de tempo, é necessario a utilização de uma pool de impressão.

Uma pool de impressão permite que várias impressoras físicas possam ser controladas por uma única impressora lógica, sempre que um trabalho for enviado para a impressora lógica, esta consultará o estado das impressoras físicas para verificar qual equipamento está livre no momento e enviará o trabalho para ela . Um exemplo de pool de impressão é demostrada na figura 1.

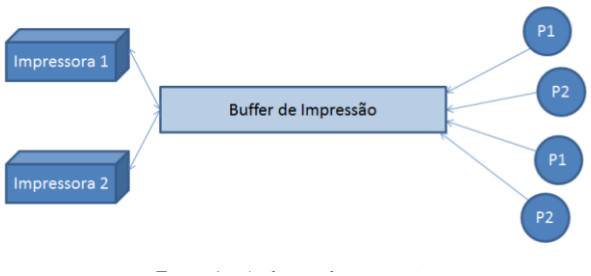

Figura 1. Proposta do exercício computacional.

2. Fundamentação Téorica

Nesta sessão serão tradatas, as tecnologias que foram utilizadas neste documento.

2.1. Socket

Segundo [2].[3]. Os sockets são compostos por um conjunto de primitivas do sistema operacional e foram originalmente desenvolvidos para o BSD Unix. Podem ser utilizados nos mais variados sistemas operacionais com recursos de comunicação em rede, sendo suportados pela maioria das linguagens de programação. Sockets são suportados em Java desde o JDK 1.0, para sua utilização devemos fazer uso das classes contidas no pacote

java.net. Um exemplo interessante da programação de sockets em Java são os drivers JDBC do tipo 4, que usam sockets para comunicar-se diretamente com a API de rede do banco de dados. O fluxo de troca de dados de um socket pode ser observada na figura 2.

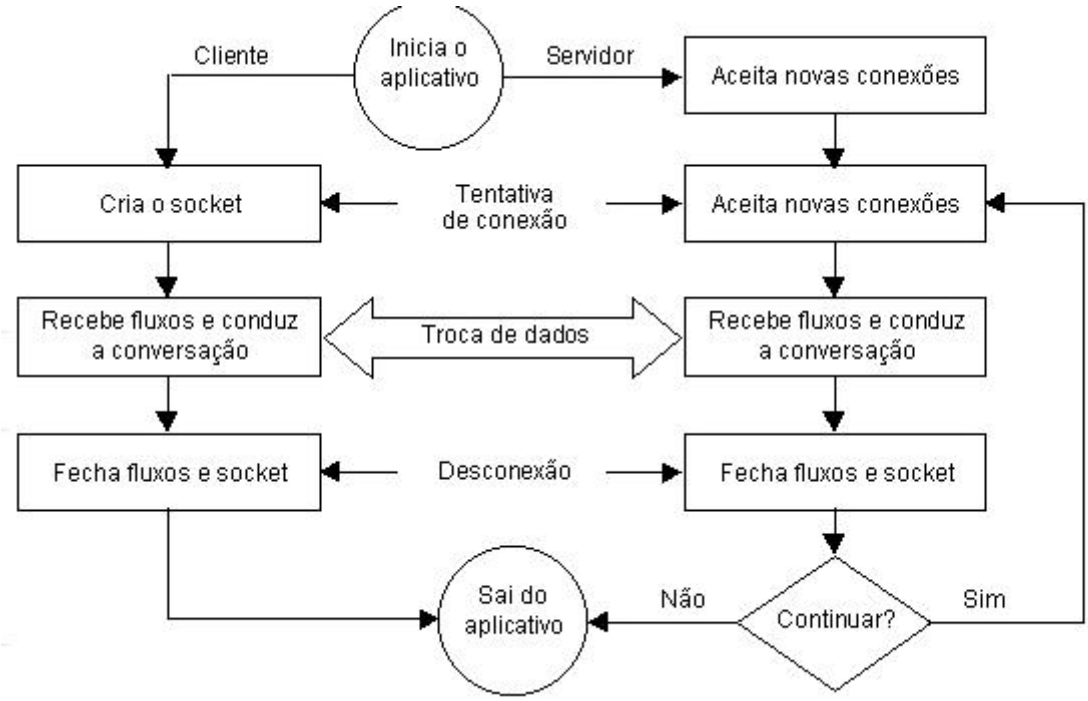

Figura 2. Fluxo de troca de dados com sockets.

2.2. Problema de Concorrência

Essa ideia é chamada de região crítica. É um pedaço de código que definimos como crítico e que não pode ser executado por duas threads ao mesmo tempo. Apenas uma thread por vez consegue entrar em alguma região crítica.

3. Desenvolvimento

Para resolver o trabalho proposto, foi desenvolvido uma aplicação java que simula, uma pool de impressões que as ordena e envia para uma impressora. Foram criadas 3 classes Javas sendo elas: Cliente , Pool de Impressão e Impressora. As mesmas serão comentandas nas póximas sessões.

3.1. Cliente

Essa classe se conecta a classe servidor, repassando informações via Socket, em uma via de mão dupla. Os dados serão aceitos na memória e posteriormente serão alocados no pelo escalonardo da pool de impressão, em alguma impressora disponível, onde mais tarde será notificado com o resultado da processamento da impressão.

O código responsável pela chamda de impressão no cliente e simples, podemos observar o método na figura 3.

```java
public void executa() throws UnknownHostException, IOException, InterruptedException {
    System.out.println("Bem vindo digite algo para ser impresso!");
    BufferedReader rf = new BufferedReader(new InputStreamReader(System.in));
    String mensagem = rf.readLine();
    Socket cliente = new Socket(this.host, this.porta);
    System.out.println("O cliente de numero "+ cliente.getLocalPort() + " se conectou ao servidor "+ cliente.getPort()+"!");

    Recebedor r = new Recebedor(cliente.getInputStream());
    Thread t1 = new Thread(r);
    t1.start();
    PrintStream saida = new PrintStream(cliente.getOutputStream());

    saida.println(mensagem);

    t1.join();
    saida.close();
    cliente.close();
}
```

Figura 3. Código cliente, pool de impressão.

3.2. Pool de Impressão - Servidor

Classe responsável pela comunicação dos Sockets (Cliente/Impressora), ela contém o escalonador de impressão, que é responsável por designar os serviços na pool de impressão.

O código do servidor é responsável por 3 threads sendo elas:

1. Aceitar conexões via Socket e adicionar documento para impressão.
2. Procurar impressoras disponíveis (escalonador de impressão)
3. Enviar documento para impressoras e receber resposta de confirmação das mesmas.

A figura 4 demostra a função que inicializa as variáveis proposta no exercício computacional, além de iniciar as Threads do servidor.

```java
public void executa() throws IOException {
    BufferedReader rd = new BufferedReader(new InputStreamReader(System.in));
    System.out.println("#### BEMVINDO AO POOL DE IMPRESSÃO RAFAEL VIANA ####");
    System.out.println("Informe em sequência R M T separados por espaço entre os valores!");
    String[] linha = rd.readLine().split( regex: " ");
    Buffer.setT(Integer.parseInt(linha[2]));
    Buffer.setM(Integer.parseInt(linha[1]));
    Buffer.setR(Integer.parseInt(linha[0]));
    System.out.println("Servidor Rodando na Porta "+ porta+"!");
    Buffer.addImpressora(new Impressora( ip: "127.0.0.1", porta: 2323, name: "2323"));
    Buffer.addImpressora(new Impressora( ip: "127.0.0.1", porta: 2324, name: "2324"));
    SaidaDocumento escalonador = new SaidaDocumento();
    new Thread(escalonador).start();
    ServerSocket servidor = new ServerSocket(this.porta);
    while (true) {
        Socket cliente = servidor.accept();
        ChegadaDocumento tc = new ChegadaDocumento(cliente, new Scanner(cliente.getInputStream()).nextLine(),
                new PrintStream(cliente.getOutputStream()));
        new Thread(tc).start();
    }
}
```

Figura 4.

O código completo, incluindo todas as classes como na figura 5, estão disponíveis no link https://github.com/rafaelgov95/SD/tree/master/PTJI/Projeto-PTJ

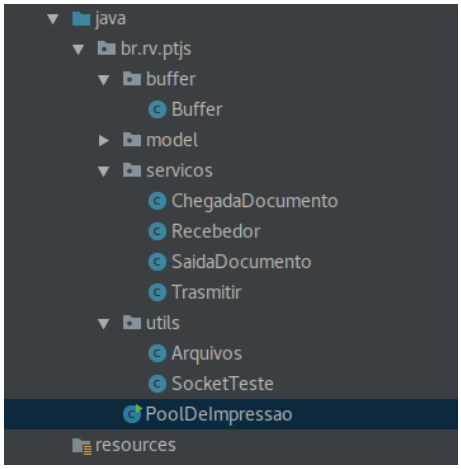

Figura 5. Classes do Servidor de pool de Impressão

3.3. Impressoras

Classe responsável pela impressão dos documentos contidos no buffer da Pool de Impressão 3.2.

Ela trabalha junto ao escalonador do servidor, que é responsável por notificada as impressoras cadastradas, que novas impressões devem ser executadas.

A figura 6, apresenta o método que realiza a criação das Threads. Elas executam em modo servidor aguardando o escalonador alocar alguma tarefa.

```
public void executa() throws IOException, InterruptedException {
    StartImpressora imp1 = new StartImpressora( porta: 2323);
    StartImpressora imp2 = new StartImpressora( porta: 2324);
    Thread t1 = new Thread(imp1);
    t1.start();
    Thread t2 = new Thread(imp2);
    t2.start();
    t1.join();
    t2.join();

}
```

Figura 6. Criação de 2 impressoras pelo método StartImpressora.

Após acriação, as Threads que representam as impressoras, ficam aguardando no while(), pelo método servidor.accept como na figura 7.

```
Thread.sleep(r.nextInt( bound: 2000));
Socket cliente = servidor.accept();
```

Figura 7.

O código do laço completo e demostrado na figura 8.

```
while (true) {

    Thread.sleep(r.nextInt( bound: 2000));
    Socket cliente = servidor.accept();
    PrintStream ps = new PrintStream(cliente.getOutputStream());
    String localPort = Port(cliente);
    Thread.sleep(r.nextInt( bound: 100));
    int proba = 6;
    String status ;
    if (proba > r.nextInt( bound: 10)) {
        status=" ERRO";
        ps.println("ERRO");
    } else {
        status="  OK";
        ps.println("OK");
    }
    if(!localPort.equals("Estralho")) {
        String log = "Impressão: " +(cont++)+" Cliente " + localPort + " Data: " + LocalTime.now().toString()+status;
        Arquivos.CriarArquioX( nome: "logs_" + porta,  caminho: "./Logs", log);
    }


    ps.close();
    cliente.close();
}
```

Figura 8.

4. Testes e Resultados

Nesta sessão será demostrado os relatórios após a conexão de 4 clientes, com o servidor e posteriormente a distribuição dos documentos entre as impressoras disponíveis. Após o processamento uma resposta de impressão é encaminhada para os clientes. O relatório do escalonador, após a execução dos 4 clientes é demostrado na figura 9.

```
|Cliente       13144     | Data: 21:13:39.680474|
|Cliente       13150     | Data: 21:13:40.591087|
|Impressora 2323| Cliente 13144| Data: 21:13:41.275060| ERRO
|Impressora 2324| Cliente 13150| Data: 21:13:41.648121| ERRO
|Cliente       13158     | Data: 21:13:42.459285|
|Impressora 2323| Cliente 13144| Data: 21:13:43.288844| ERRO
|Impressora 2323| Cliente 13144| Data: 21:13:43.505711| ERRO
|Impressora 2324| Cliente 13150| Data: 21:13:43.671764| OK
|Cliente       13162     | Data: 21:13:44.454779|
|Impressora 2323| Cliente 13144| Data: 21:13:44.790730| ERRO
|Impressora 2323| Cliente 13144| Data: 21:13:44.980497| ERRO
|Impressora 2324| Cliente 13158| Data: 21:13:46.659684| ERRO
|Impressora 2324| Cliente 13158| Data: 21:13:46.786009| ERRO
|Impressora 2323| Cliente 13144| Data: 21:13:46.937070| ERRO
|Impressora 2324| Cliente 13158| Data: 21:13:47.854170| ERRO
|Impressora 2324| Cliente 13158| Data: 21:13:48.304046| ERRO
|Impressora 2323| Cliente 13144| Data: 21:13:48.681945| OK
|Impressora 2324| Cliente 13158| Data: 21:13:50.061893| ERRO
|Impressora 2323| Cliente 13162| Data: 21:13:50.605932| ERRO
|Impressora 2324| Cliente 13158| Data: 21:13:51.008731| ERRO
|Impressora 2323| Cliente 13162| Data: 21:13:51.255589| ERRO
|Impressora 2323| Cliente 13162| Data: 21:13:52.341388| OK
|Impressora 2324| Cliente 13158| Data: 21:13:52.975600| OK
```

Figura 9. Relatório de impressões, registrado pelo escalonador presente no servidor.

O relatório da impressora 2323, pode ser observado na figura 10.

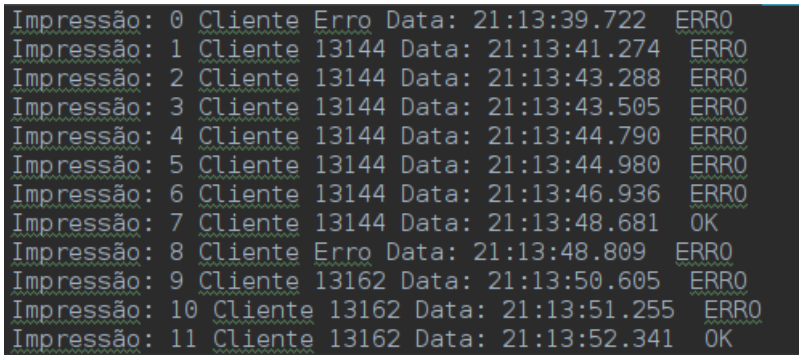

Figura 10. Relatório da impressora 2323, após execução dos 4 clientes.

O relatório da impressora 2324, pode ser observado na figura 11.

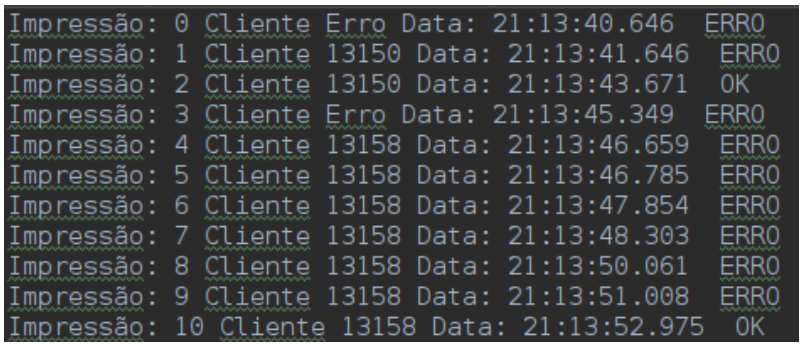

Figura 11. Relatório da impressora 2324, após execução dos 4 clientes.

O cliente após o envio do documento, aguarda a impressão do mesmo, enquanto isso recebe notificações a respeito de sua impressão em tempo real, podemos ver na figura 12, o resultado final de notificações do cliente.

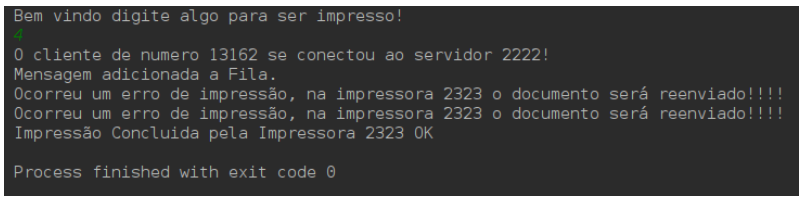

Figura 12. Notificações em tempo real da impressão para o Cliente.

5. Conclusão

Neste relatório foi apresentado o desenvolvimento de uma pool de impressão, assim como a problematica da região crítica de mémoria compatilhada entre Threads, e a sua solução utilizando o *synchronized*. Para realizar a comunicação entre as Threads foram utilizadas conexões Sockets.

Referências

[1] E. Software, "Pool de impressão – você sabe sua finalidade?." http://www.devmedia.com.br/java-sockets-criando-comunicacoes-em-java/9465, 2017. [Online; acesso em 22-Outubro-2017].

[2] Caelum, "Apêndice - Sockets." https://www.caelum.com.br/apostila-java-orientacao-objetos/apendice-sockets/, 2017. [Online; acesso em 18-Outubro-2017].

[3] Caelum, "Apêndice - Problemas com concorrência." https://www.caelum.com.br/apostila-java-orientacao-objetos/apendice-problemas-com-concorrencia/, 2017. [Online; acesso em 20-Outubro-2017].