

Simulação de Pool de Impressão Distribuída Utilizando Socket

Exercício Computacional II - Sistemas Distribuídos

Rafael Gonçalves de Oliveira Viana¹

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

rafael.viana@aluno.ufms.br
25 de outubro de 2017

Resumo. Este documento relata como foi o desenvolvimento de uma Pool de Impressão na linguagem Java versão 8, utilizando Threads e Sockets.

1. Introdução

De acordo com [1], em um ambiente de trabalho, onde desejasse imprimir documentos enviados, a um curto período de tempo, é necessário 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 demonstrada na figura 1.



Figura 1. Proposta do exercício computacional.

O objetivo deste relatório é expor as dificuldades e descobertas que a implementação de uma pool de impressão, ofertou para a vida acadêmica.

2. Fundamentação Teórica

Nesta seção serão tratadas, soluções de problemas pertinentes que foram utilizadas neste documento.

2.1. Threads

Segundo [2]. Cada thread é processado de forma aparentemente simultânea, uma vez que a mudança entre um segmento e outro é feita tão rapidamente que para o usuário, isso está acontecendo em paralelo. Em hardware multi-CPU ou multi-cores, os threads são realizados de forma simultânea. Os sistemas que suportam um único segmento (na execução real) são chamados monothread, enquanto os sistemas que suportam múltiplos threads são chamados multithread.

2.2. Problema de Concorrência

O problema de concorrência existem em diversos cenários em que, um pedaço de código que definimos como crítico não pode ser executado por duas threads ou mais, ao mesmo tempo. Apenas uma thread por vez consegue entrar em alguma região crítica e realizar determinada função.

O java realiza a sincronização das threads colocando a palavra reservada `synchronized` nos métodos ou utilizando um objeto como parâmetro `synchronized(Objeto)`.

Para mais informações referente a sincronização de threads na linguagem java consultar a referência [3].

2.3. Socket

Segundo [4],[5]. Sockets são estruturas que permitem que funções de software se interconectem. O conceito é o mesmo de um soquete (elétrico, telefônico, etc ...), que serve para interconectar elementos diferentes. O termo é usado para especificar uma estrutura que faz com que rotinas de software na mesma máquina ou em máquinas diferentes possam se comunicar. Mais especificamente, o termo é usado para softwares usados na internet, para especificar a conexão entre o software da camada de aplicação (protocolos HTTP, FTP, TELNET, POP3, etc...) e a camada de transporte (protocolos TCP ou UDP).

Os sockets são compostos por um conjunto de primitivas do sistema operacional e foram originalmente desenvolvidos para o BSD Unix. Sockets são suportados em Java desde o JDK 1.0, para sua utilização devemos fazer uso das classes contidas no pacote `java.net`. Podemos observar na figura 2, o funcionamento de dois tipos de Sockets *Servidor* e *Cliente*.

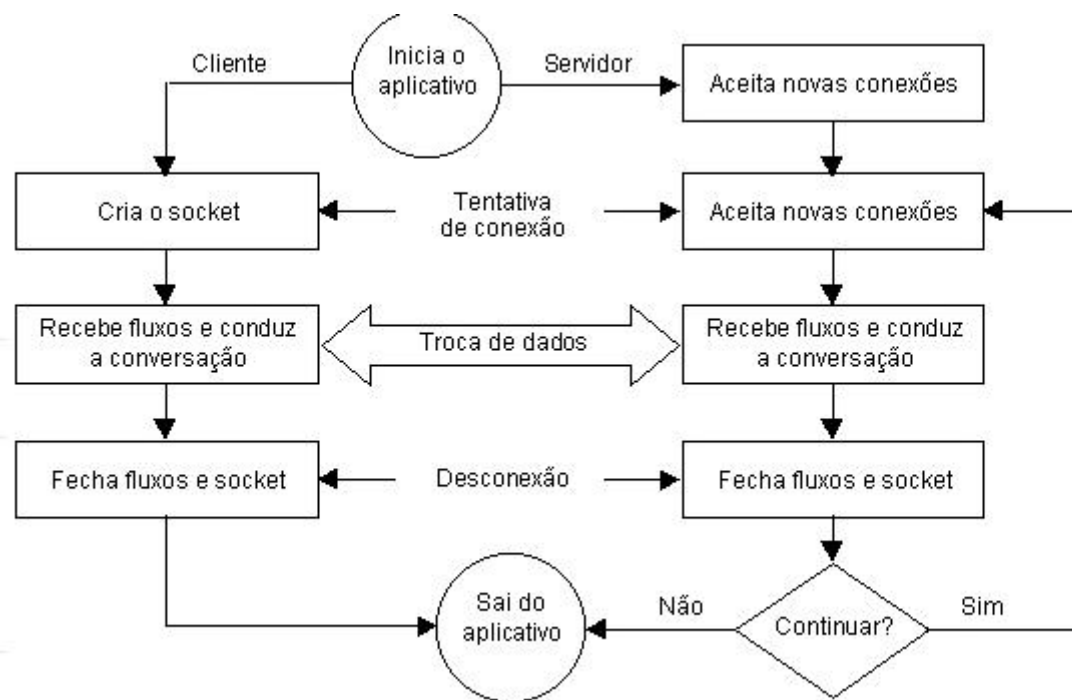


Figura 2. Fluxo de troca de dados com sockets.

2.4. Problema de Espera Ocupada

Para mais informações referente a espera ocupada de threads na linguagem java consultar as referências [5] [6].

3. Desenvolvimento

Para resolver o trabalho proposto, no qual 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 comentadas nas próximas seções.

3.1. Cliente

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

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

```

public class Cliente {
    public static void main(String[] args)
        throws UnknownHostException, IOException, InterruptedException {
        new Cliente( host: "sy.gov.dgns.net", porta: 3000).executa();
    }

    private String host;
    private int porta;

    public Cliente(String host, int porta) {
        this.host = host;
        this.porta = porta;
    }

    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, da 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 demonstra a função que inicializa as variáveis proposta no exercício computacional, além de iniciar as Threads do servidor. O responsável por determinar quais impressoras estão disponíveis, e alocar trabalho para as mesmas, é o escalonador de impressão que é desenvolvido na classe *Chegada de Documento*. O escalonador ao perceber que não tem tarefas a realizar vai dormir wait(), quando uma mensagem chega para ser escalonada, ela acorda o escalonador, e ele realiza o repasse para a primeira impressora disponível, e volta a dormir.

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 notificar as impressoras cadastradas, que novas impressões devem ser executadas, vale destacar que o escalonador foi desenvolvido, de acordo com a boa prática de não deixar uma espera ocupada.

A figura 5, 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 {
    BufferedReader rd = new BufferedReader(new InputStreamReader(System.in));

    System.out.println("#### BEM VINDO AO POOL DE IMPRESSÃO RAFAEL VIANA ####\n" +
        "Exemplos de valores de entrada\n"+
        "R = (0 a 100), 0 representando 0% e 100 representando 100% de probabilidade de erro de impressão.\n" +
        "M = (1 a N>1) Valor inteiro do Tamanho Buffer do Servidor.\n" +
        "T = (1 a 3) Valor inteiro\n" +
        "Informe em sequência R M T , separados por espaço entre os valores!");

    String[] linha = rd.readLine().split(" ");
    Buffer b = new Buffer(Integer.parseInt(linha[0]), Integer.parseInt(linha[1]), Integer.parseInt(linha[2]));
    System.out.println("Servidor Rodando na Porta "+ porta+ "!");
    b.addImpressora(new Impressora( ip: "sdufms2017.ddns.net", porta: 4200, name: "4200"));
    b.addImpressora(new Impressora( ip: "sdufms2017.ddns.net", porta: 4300, name: "4300"));
    SaidaDocumento escalonador = new SaidaDocumento(b);
    Thread esc = new Thread(escalonador);
    esc.start();
    ServerSocket servidor = new ServerSocket(this.porta);
    while (true) {
        Socket cliente = servidor.accept();
        ChegadaDocumento tc = new ChegadaDocumento(b, cliente, new Scanner(cliente.getInputStream()).nextLine(),
            new PrintStream(cliente.getOutputStream()));
        new Thread(tc).start();
    }
}

```

Figura 4.

```

public void executa() throws IOException, InterruptedException {
    StartImpressora imp1 = new StartImpressora( porta: 4200);
    StartImpressora imp2 = new StartImpressora( porta: 4300);
    Thread t1 = new Thread(imp1);
    t1.start();
    Thread t2 = new Thread(imp2);
    t2.start();
    t1.join();
    t2.join();
}

```

Figura 5. Criação de duas impressoras pelo método StartImpressora.

Após a criação, as Threads que representam as impressoras, as mesmas ficam aguardando no while(true), que não gera espera ocupada, por causa do método de servidor, *servidor.accept*, como na figura 6.

4. Visão geral da implementação

Nesta seção será apresnetado os comandos para poder utilizar o Pool de Impressão.

4.1. Cliente

O cliente já está configurado para porta 3000, com endereço svgov.ddns.net que, redireciona para o endereço IP dinâmico da Oi Velox residencial, onde está a pool de impressão.

4.2. Servidor

O servidor de pool, já possui suas impressoras configuradas nas portas 4200 e 4300 com endereço sdufms2017.ddns.net, que redireciona para o endereço IP da virtual machine da Google. Quando o servidor inicia ele necessita de 3 parametros, R (Porcentagem de erros que vai de 0% a 100%), M (Tamanho do Buffer) e T (Milisegundos Default).

```

ServerSocket servidor = new ServerSocket(this.porta);
Random r = new Random();
System.out.println("Entro: " + this.porta);
while (true) {
    if (r.nextInt( bound: 100) < 15) {
        String log = "Impressora " + this.porta + " está Off-line";
        Arquivos.CriarArquivoX( nome: "logs_" + porta, caminhos: "./Logs", log);
        Thread.sleep( millis: r.nextInt( bound: 2000) + 1000);
        log = "Impressora " + this.porta + " está On-line";
        Arquivos.CriarArquivoX( nome: "logs_" + porta, caminhos: "./Logs", log);
        System.out.println("Volto: " + this.porta);
    }
    Socket cliente = servidor.accept();
    PrintStream ps = new PrintStream(cliente.getOutputStream());
    String localPort = Port(cliente);
    String status;
    if (r.nextInt( bound: 100) < 15) {
        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.CriarArquivoX( nome: "logs_" + porta, caminhos: "./Logs", log);
    }
}

```

Figura 6.

4.3. Impressoras

As impressoras já estão configuradas como Sockets servidores, aguardando mensagens encaminhadas para o endereço IP `sdufms2017.ddns.net` nas portas 4200 e 4300.

5. Testes e Resultados

Nesta seção será demonstrado os logs das impressoras e do servidor após a conexão de 40 clientes, com o servidor. Para realizar os testes foram utilizado um Raspberry Pi3 e uma Virtual Machine da Google Developers.

5.0.1. Distribuição de Recursos

1. Como servidor da pool de impressão ficou o Raspberry Pi3, demonstrando que a classe Servidor não precisa de um grande poder computacional para ser executada.
2. Como impressoras virtuais da pool de impressão, que se encontra no Raspberry Pi3, foi utilizado uma Virtual Machine da Google Developer, executando o código de impressoras virtuais relatado nesse artigo.

5.0.2. Passos

1. Primeiramente a requisição de algum cliente, se conecta ao pool de impressão, presente no Raspberry Pi3, por meio de um Socket.
2. Ao receber a requisição de impressão, a pool adiciona o documento ao Buffer, e acorda o escalonador de impressão que envia a impressão para, alguma impressora disponível na virtual machine da Google developers.
3. Ao receber o documento a impressora responde um "OK" para o pool, porém ela pode perder o documento e então envia uma mensagem "ERRO", o que provocaria o reenvio da mesma, até que consiga realizar a impressão.
4. Durante todos esses passos mensagens como "Adicionada à Fila" ou mensagens de erros, são encaminhadas para os clientes em tempo real.

Agora que já entendemos como funciona a comunicação, podemos visualizar os relatórios. Começaremos pelo do escalonador, que após a execução dos 40 clientes tem um log equivalente ao demonstrado na figura 7.

Cliente IP:177.79.78.83	Porta:15422	Data: 16:24:03.214	
A Impressora 4200 concluiu, a impressão do	Cliente IP:ip-177-79-78-83.user.vivozap.com.br Porta TCP:15422	Data: 16:24:03.999	OK Duração em Milissegundos: 778
Cliente IP:177.79.78.83	Porta:15420	Data: 16:24:05.353	
A Impressora 4300 concluiu, a impressão do	Cliente IP:ip-177-79-78-83.user.vivozap.com.br Porta TCP:15420	Data: 16:24:05.893	OK Duração em Milissegundos: 541
Cliente IP:177.79.31.54	Porta:5463	Data: 16:24:07.267	
A Impressora 4200 concluiu, a impressão do	Cliente IP:ip-177-79-31-54.user.vivozap.com.br Porta TCP:5463	Data: 16:24:07.707	OK Duração em Milissegundos: 442
Cliente IP:177.79.78.83	Porta:15413	Data: 16:24:09.944	
A Impressora 4300 concluiu, a impressão do	Cliente IP:ip-177-79-78-83.user.vivozap.com.br Porta TCP:15413	Data: 16:24:10.862	ERRO AO ENCAMINHAR
A Impressora 4300 concluiu, a impressão do	Cliente IP:ip-177-79-78-83.user.vivozap.com.br Porta TCP:15413	Data: 16:24:11.086	OK Duração em Milissegundos: 1143
Cliente IP:177.79.31.54	Porta:5443	Data: 16:24:12.311	
A Impressora 4200 concluiu, a impressão do	Cliente IP:ip-177-79-31-54.user.vivozap.com.br Porta TCP:5443	Data: 16:24:12.747	OK Duração em Milissegundos: 437
Cliente IP:177.79.31.54	Porta:5451	Data: 16:24:14.372	
A Impressora 4300 concluiu, a impressão do	Cliente IP:ip-177-79-31-54.user.vivozap.com.br Porta TCP:5451	Data: 16:24:14.828	OK Duração em Milissegundos: 457
Cliente IP:177.79.78.83	Porta:15419	Data: 16:24:18.475	
A Impressora 4200 concluiu, a impressão do	Cliente IP:ip-177-79-78-83.user.vivozap.com.br Porta TCP:15419	Data: 16:24:18.911	OK Duração em Milissegundos: 437
Cliente IP:177.79.31.54	Porta:5450	Data: 16:24:20.521	
A Impressora 4300 concluiu, a impressão do	Cliente IP:ip-177-79-31-54.user.vivozap.com.br Porta TCP:5450	Data: 16:24:20.977	OK Duração em Milissegundos: 457
Cliente IP:177.79.31.54	Porta:5461	Data: 16:24:25.432	
A Impressora 4200 concluiu, a impressão do	Cliente IP:ip-177-79-31-54.user.vivozap.com.br Porta TCP:5461	Data: 16:24:25.869	OK Duração em Milissegundos: 438
Cliente IP:177.79.78.83	Porta:15398	Data: 16:24:27.488	
A Impressora 4300 concluiu, a impressão do	Cliente IP:ip-177-79-78-83.user.vivozap.com.br Porta TCP:15398	Data: 16:24:27.933	OK Duração em Milissegundos: 446
Cliente IP:177.79.31.54	Porta:5471	Data: 16:24:29.516	
A Impressora 4200 concluiu, a impressão do	Cliente IP:ip-177-79-31-54.user.vivozap.com.br Porta TCP:5471	Data: 16:24:30.065	OK Duração em Milissegundos: 550
Cliente IP:177.79.31.54	Porta:5455	Data: 16:24:31.566	
A Impressora 4300 concluiu, a impressão do	Cliente IP:ip-177-79-31-54.user.vivozap.com.br Porta TCP:5455	Data: 16:24:32.282	OK Duração em Milissegundos: 637
Cliente IP:177.79.78.83	Porta:15406	Data: 16:24:34.245	
A Impressora 4200 concluiu, a impressão do	Cliente IP:ip-177-79-78-83.user.vivozap.com.br Porta TCP:15406	Data: 16:24:34.749	OK Duração em Milissegundos: 496
Cliente IP:177.79.78.83	Porta:15402	Data: 16:24:36.290	
A Impressora 4300 concluiu, a impressão do	Cliente IP:ip-177-79-78-83.user.vivozap.com.br Porta TCP:15402	Data: 16:24:36.737	OK Duração em Milissegundos: 442
Cliente IP:177.79.78.83	Porta:15400	Data: 16:24:39.737	
A Impressora 4200 concluiu, a impressão do	Cliente IP:ip-177-79-78-83.user.vivozap.com.br Porta TCP:15400	Data: 16:24:40.187	OK Duração em Milissegundos: 451

Figura 7. Relatório de impressões, registrado pelo escalonador presente no servidor.

```

Impressão: 0 POOL TCP:46652 Data: 21:24:04.076 OK
Impressão: 1 POOL TCP:46654 Data: 21:24:04.304 OK
Impressão: 2 POOL TCP:46660 Data: 21:24:07.588 ERRO
Impressão: 3 POOL TCP:46662 Data: 21:24:08.029 OK
Impressão: 4 POOL TCP:46670 Data: 21:24:12.626 OK
Impressão: 5 POOL TCP:46672 Data: 21:24:13.062 OK
Impressão: 6 POOL TCP:46678 Data: 21:24:18.793 ERRO
Impressão: 7 POOL TCP:46680 Data: 21:24:19.239 OK
Impressão: 8 POOL TCP:46686 Data: 21:24:25.751 OK
Impressão: 9 POOL TCP:46688 Data: 21:24:26.177 OK
Impressão: 10 POOL TCP:46694 Data: 21:24:29.833 OK
Impressão: 11 POOL TCP:46696 Data: 21:24:30.369 OK
Impressão: 12 POOL TCP:46702 Data: 21:24:34.622 OK
Impressão: 13 POOL TCP:46704 Data: 21:24:35.049 OK
Impressão: 14 POOL TCP:46710 Data: 21:24:40.057 OK
Impressão: 15 POOL TCP:46712 Data: 21:24:40.496 OK
Impressão: 16 POOL TCP:46718 Data: 21:24:45.014 OK
Impressão: 17 POOL TCP:46720 Data: 21:24:45.457 OK
Impressão: 18 POOL TCP:46726 Data: 21:24:49.514 OK
Impressão: 19 POOL TCP:46728 Data: 21:24:49.936 OK
Impressão: 20 POOL TCP:46734 Data: 21:24:53.625 OK
Impressão: 21 POOL TCP:46736 Data: 21:24:54.042 OK
Impressão: 22 POOL TCP:46742 Data: 21:24:57.688 OK
Impressão: 23 POOL TCP:46744 Data: 21:24:58.111 OK
Impressão: 24 POOL TCP:46750 Data: 21:25:02.822 OK
Impressão: 25 POOL TCP:46752 Data: 21:25:03.252 OK
Impressão: 26 POOL TCP:46760 Data: 21:25:08.559 OK
Impressão: 27 POOL TCP:46762 Data: 21:25:08.976 OK
Impressora 4200 está Off-line | Data: 21:25:08.976
Impressora 4200 está On-line | Data: 21:25:11.312
Impressão: 28 POOL TCP:46772 Data: 21:25:13.689 OK
Impressão: 29 POOL TCP:46774 Data: 21:25:14.115 ERRO

```

(a) Impressora 4200

```

Impressão: 0 POOL TCP:58790 Data: 21:24:05.664 OK
Impressão: 1 POOL TCP:58792 Data: 21:24:06.210 OK
Impressão: 2 POOL TCP:58798 Data: 21:24:10.263 OK
Impressão: 3 POOL TCP:58800 Data: 21:24:10.692 ERRO
Impressão: 4 POOL TCP:58802 Data: 21:24:11.403 OK
Impressão: 5 POOL TCP:58808 Data: 21:24:14.687 ERRO
Impressora 4300 está Off-line | Data: 21:24:14.688
Impressora 4300 está On-line | Data: 21:24:17.212
Impressão: 6 POOL TCP:58810 Data: 21:24:17.213 OK
Impressão: 7 POOL TCP:58816 Data: 21:24:20.848 OK
Impressão: 8 POOL TCP:58818 Data: 21:24:21.298 OK
Impressão: 9 POOL TCP:58824 Data: 21:24:27.809 OK
Impressão: 10 POOL TCP:58826 Data: 21:24:28.248 OK
Impressão: 11 POOL TCP:58832 Data: 21:24:31.881 OK
Impressão: 12 POOL TCP:58834 Data: 21:24:32.513 OK
Impressão: 13 POOL TCP:58840 Data: 21:24:36.614 ERRO
Impressão: 14 POOL TCP:58842 Data: 21:24:37.071 OK
Impressão: 15 POOL TCP:58848 Data: 21:24:42.416 OK
Impressão: 16 POOL TCP:58850 Data: 21:24:42.842 OK
Impressão: 17 POOL TCP:58856 Data: 21:24:47.702 OK
Impressão: 18 POOL TCP:58858 Data: 21:24:48.122 OK
Impressão: 19 POOL TCP:58864 Data: 21:24:51.755 OK
Impressão: 20 POOL TCP:58866 Data: 21:24:52.282 OK
Impressão: 21 POOL TCP:58872 Data: 21:24:55.601 OK
Impressão: 22 POOL TCP:58874 Data: 21:24:56.113 OK
Impressão: 23 POOL TCP:58880 Data: 21:24:59.791 OK
Impressora 4300 está Off-line | Data: 21:24:59.791
Impressora 4300 está On-line | Data: 21:25:01.182
Impressão: 24 POOL TCP:58882 Data: 21:25:01.182 OK
Impressão: 25 POOL TCP:58888 Data: 21:25:05.320 OK

```

(b) Impressora 4300

Figura 8. Relatório parcial das impressoras.

Os relatórios das impressoras 4200 e 4300, pode ser observado na figura 8. 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 9, o resultado final de notificações do cliente.

```

0 cliente IP:192.168.43.165 Porta TCP:22920 se conectou ao servidor IP:201.25.83.247 TCP: 3000!
Adicionada à Fila.
Ocorreu um erro, ao encaminha para impressora 4200, uma nova tentativa será realizada !!!!
Impressão Concluída pela Impressora 4200 em 1738 Milissegundos OK

```

Figura 9. Notificações em tempo real de impressão para o Cliente.

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

6. Conclusão

Neste relatório foi demonstrado o desenvolvimento de uma pool de impressão, enfrentando problemáticas como: região crítica de memória compartilhada entre Threads, comunicação via Sockets geograficamente distantes, espera ocupada do escalonadores, e apontando suas soluções utilizando o *synchronized*, *SocketJava* e *Wait()/Notify* respectivamente.

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] Wikipedia, “Threads.” [https://pt.wikipedia.org/wiki/Thread_\(ci%C3%A7%C3%A3o_da_computa%C3%A7%C3%A3o\)](https://pt.wikipedia.org/wiki/Thread_(ci%C3%A7%C3%A3o_da_computa%C3%A7%C3%A3o)), 2017. [Online; acesso em 22-Outubro-2017].
- [3] Jacques, “Sincronização de Threads.” <http://www.dsc.ufcg.edu.br/~jacques/cursos/map/html/threads/sincronizacao.html>, 2017. [Online; acesso em 22-Outubro-2017].
- [4] Caelum, “Apêndice - Sockets.” <https://www.caelum.com.br/apostila-java-orientacao-objetos/apendice-sockets/>, 2017. [Online; acesso em 18-Outubro-2017].
- [5] 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].
- [6] MC514–Sistemas, “Produtores e Consumidores.” <http://www.ic.unicamp.br/~islene/mc514/prod-cons/prod-cons.pdf>, 2017. [Online; acesso em 22-Outubro-2017].