

Trabalho Prático - 2

Francisco J. Lucca, Marlon Varoni, João Almeida

27 de outubro de 2021

1 Introdução

Neste relatório será descrita a implementação e demonstração de resultados obtidos sob o trabalho prático 2, estipulado pelo professor Sérgio Johanm. Este trabalho tem como objetivo aprofundar o conhecimento relacionado aos tópicos abordados até o momento, que incluem o modelo cliente/servidor, RPC, RMI, P2P e comunicação coletiva.

A tarefa consiste na implementação de um sistema P2P que deve ser organizado como uma arquitetura híbrida, onde o controle de toda a aplicação (lógica e estado) é concentrado em um grupo de supernodos (pelo menos 3). Um único programa deve ser utilizado, e o mesmo pode ser configurado em um dos dois modos de operação (supernodo/nodo P2P). Para isso, pode-se passar essa informação como parâmetro durante a carga do programa, juntamente com outras informações de configuração.

2 Organização do Código

A organização do código utiliza um sistema *P2P* com uma arquitetura híbrida. O controle da aplicação é feito por um grupo de nodos (denominados *Supernodos*) que comunicam-se entre si através de *Multicast*. O mesmo código serve também para a instanciação dos Nodos P2P, que entrarão no grupo e disponibilizarão seus recursos.

Uma vez que o nodo entra no grupo, conecta-se diretamente com um supernodo através de IP e porta. Feito isso, ele adiciona todos os seus recursos na lista de recursos, controla pelos supernodos. Após isso um nodo pode solicitar a lista de recursos, perguntar aos supernodos em qual IP o recurso X está e contatar outro nodo do grupo para recuperar o recurso.

Abaixo seguem explicações mais detalhadas sobre o funcionamento de cada classe criada.

2.1 Enum Tags

O *Enum* define, assim como o nome sugere, as *tags* que os nodos e supernodos utilizam para se comunicar. As *tags* criadas foram:

```
SUPERNODO_REQUEST_RECURSOS ,  
SUPERNODO_REQUEST_RECURSO ,  
REGISTRA_NOVO_NODO ,  
NODO_REQUEST_RECURSOS ,  
NODO_RESPONSE_RECURSOS ,  
NODO_REQUEST_RECURSO ,  
NODO_RESPONSE_RECURSO ,  
P2P_REQUEST ,  
P2P_RESPONSE ,  
ALIVE
```

2.2 Classe Recurso

Esta classe define a instancia de um objeto *Recurso*. A classe possui os métodos padrões de *GET* que retorna os valores de cada variável do objeto e seu método construtor. O construtor recebe três valores diferentes, dentro eles:

- *String hash*: Conteúdo do recurso que utiliza o algoritmo *SHA-256* para calcular o seu hash;
- *InetAddress ip*: Ip do nodo em que este recurso é cadastrado
- *String nomeRecurso*: Nome do recurso

2.3 Classe Socket

A classe Socket é responsável por realizar a comunicação entre os supernodos por meio de Multicast. As variáveis guardam informações como o IP do supernodo, IP do grupo Multicast, Socket implementado e tempo da última mensagem recebida. Também foi criada uma lista para armazenar o conteúdo dos nodos e outra lista que é utilizada em casos onde a informação não esta presente no supernodo, desta forma são armazenadas no *array* informações referentes aos recursos de outros supernodos.

Os métodos desta classe são responsáveis por gerenciar a comunicação entre os nodos, a função do *getRecursos* é enviar uma mensagem em Multicast para os outros supernodos e recebendo informações referentes à todos os recursos disponíveis. O método *getResource* que contém uma string informada por parâmetro possui comportamento semelhante porém retorna o endereço IP do Nodo que contém o *Resource* requisitado na *string* informada.

O último método é uma sobrescrita do método *run* da classe Thread do Java, e é responsável por controlar o envio e recebimento de mensagens dentro do grupo Multicast.

2.4 Classe App

A classe *App* implementa o método *main*. Uma verificação é feita após o usuário informar se será executado um nodo ou supernodo, caso seja um supernodo, uma conexão é iniciada ao instanciar a classe Multicast. Um novo *socket* também é gerado para comunicação na rede.

Nesta etapa são criadas duas threads, uma para realizar a conexão entre nodo e supernodo, por meio de Unicast e outra conexão entre supernodos por Multicast. O registro de um novo nodo ocorre nesta classe, caso seja verificado que é um nodo P2P, é gerado um *Keep-Alive* que informa que existe a conexão a cada 5 segundos.

A classe App também é responsável por receber os comandos do usuário, sendo possível solicitar todos os recursos ou apenas por IP. A conexão ocorre por meio da utilização do método *send* da classe *DatagramSocket* da linguagem Java.

3 Instanciando Supernodos e Nodos P2P

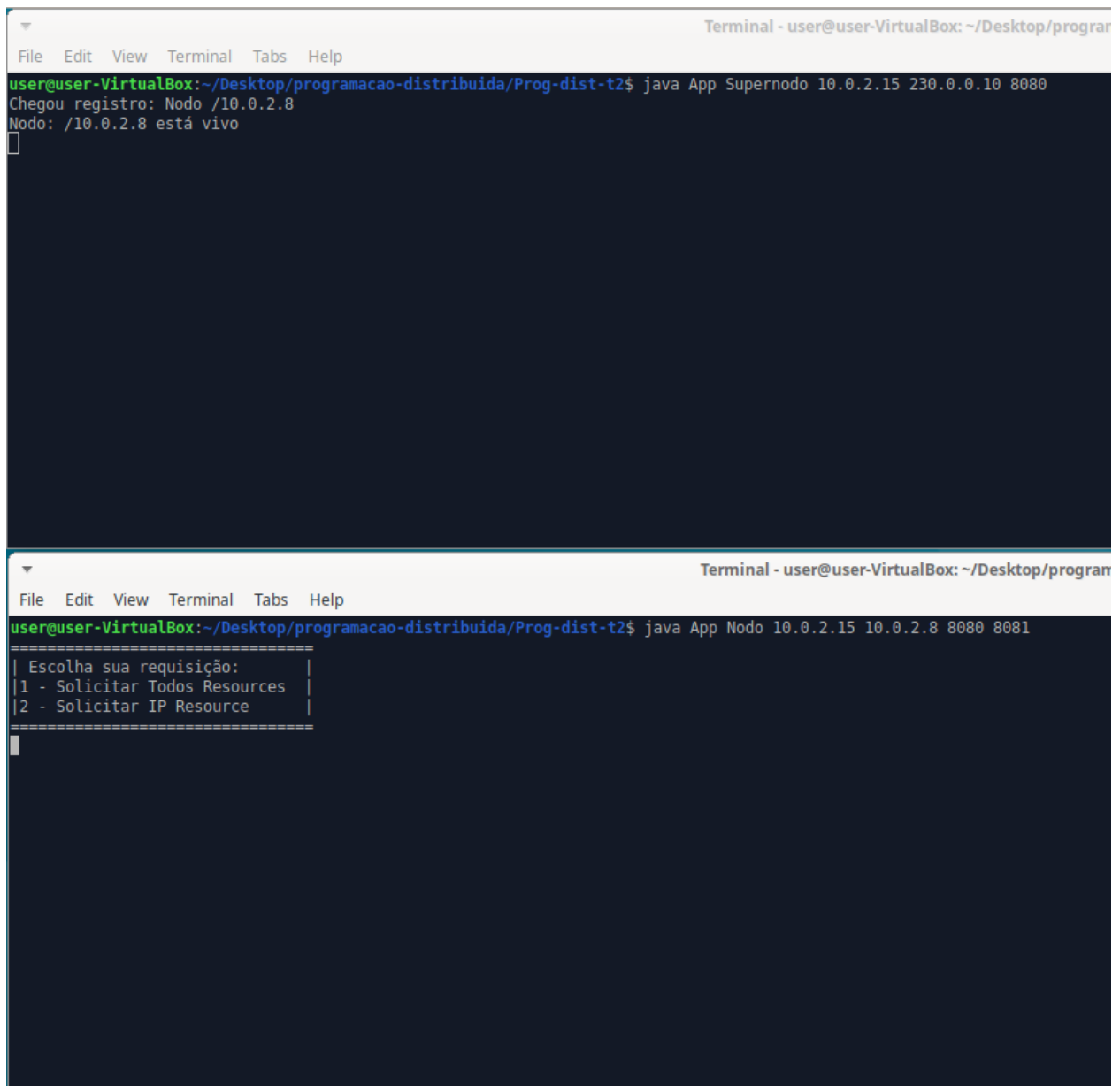
Os Nodos e Supernodos da aplicação podem ser gerados por meio da execução dos comandos abaixo:

```
$ java App Supernodo <ip_do_supernodo> <ip_do_grupo> <porta_supernodo>
```

```
$ java App Nodo <ip_do_nodo> <ip_do_supernodo> <porta_supernodo> <porta_nodo>
```

4 Demonstração da Implementação

Para a demonstração da implementação serão usadas duas maquinas virtuais, cada uma delas contendo um único arquivo. Abaixo seguem imagens da demonstração.



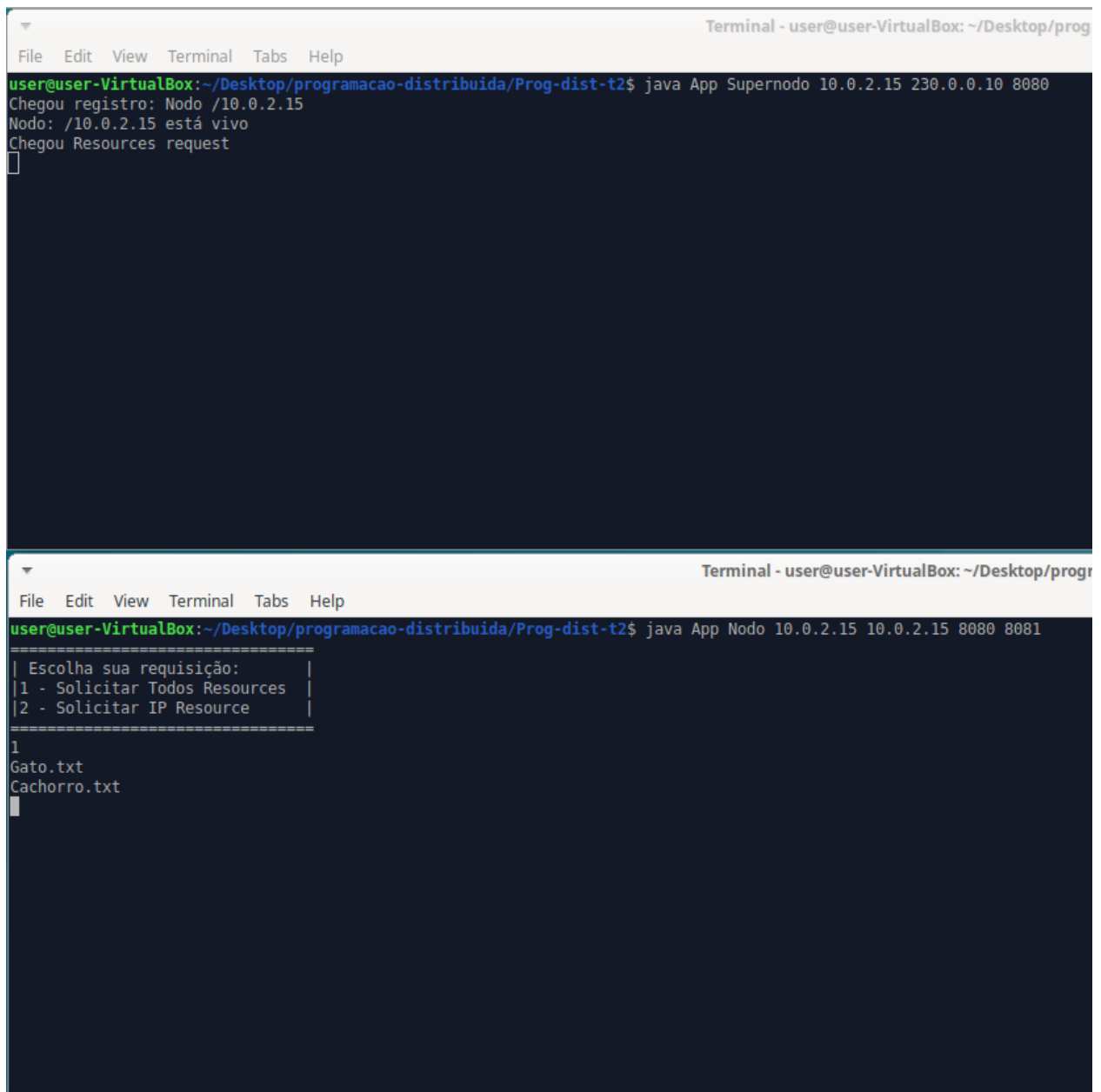
```
Terminal - user@user-VirtualBox: ~/Desktop/prograr
File Edit View Terminal Tabs Help
user@user-VirtualBox:~/Desktop/programacao-distribuida/Prog-dist-t2$ java App Supernodo 10.0.2.15 230.0.0.10 8080
Chegou registro: Nodo /10.0.2.8
Nodo: /10.0.2.8 está vivo
█

Terminal - user@user-VirtualBox: ~/Desktop/program
File Edit View Terminal Tabs Help
user@user-VirtualBox:~/Desktop/programacao-distribuida/Prog-dist-t2$ java App Nodo 10.0.2.15 10.0.2.8 8080 8081
=====
| Escolha sua requisição: |
| 1 - Solicitar Todos Resources |
| 2 - Solicitar IP Resource |
|=====
█
```

Figura 1: Configuração inicial

Na figura acima uma maquina está rodando um supernodo no endereço *10.0.2.15:8080* e um nodo que está conectado em outro supernodo, localizado no endereço *10.0.2.8:8080*.

Para o primeiro exemplo temos a solicitação de todos os recursos presentes na rede. Como cada maquina possui apenas um arquivo se espera a listagem de dois arquivos ao nodo.



```
Terminal - user@user-VirtualBox: ~/Desktop/prog
File Edit View Terminal Tabs Help
user@user-VirtualBox:~/Desktop/programacao-distribuida/Prog-dist-t2$ java App Supernodo 10.0.2.15 230.0.0.10 8080
Chegou registro: Nodo /10.0.2.15
Nodo: /10.0.2.15 está vivo
Chegou Resources request
█

Terminal - user@user-VirtualBox: ~/Desktop/prog
File Edit View Terminal Tabs Help
user@user-VirtualBox:~/Desktop/programacao-distribuida/Prog-dist-t2$ java App Nodo 10.0.2.15 10.0.2.15 8080 8081
=====
| Escolha sua requisição: |
| 1 - Solicitar Todos Resources |
| 2 - Solicitar IP Resource |
=====
1
Gato.txt
Cachorro.txt
█
```

Figura 2: Requisição de todos recursos

Para o terceiro exemplo é requerido o recurso **Cachorro.txt** e a aplicação ira retornar o *ip* de qual nodo este recurso está alocado.

```
Terminal - user@user-VirtualBox: ~/Desktop/prog
user@user-VirtualBox:~/Desktop/programacao-distribuida/Prog-dist-t2$ java App Supernodo 10.0.2.15 230.0.0.10 8080
Chegou registro: Nodo /10.0.2.15
Nodo: /10.0.2.15 está vivo
Chegou Resources request
Nodo: /10.0.2.15 está vivo
Nodo: /10.0.2.15 está vivo
Chegou Resource request
█

Terminal - user@user-VirtualBox: ~/Desktop/prog
user@user-VirtualBox:~/Desktop/programacao-distribuida/Prog-dist-t2$ java App Nodo 10.0.2.15 10.0.2.15 8080 8081
=====
| Escolha sua requisição: |
| 1 - Solicitar Todos Resources |
| 2 - Solicitar IP Resource |
=====
1
Gato.txt
Cachorro.txt
2
Qual Resource?
Cachorro.txt
Local Resource: /10.0.2.15
█
```

Figura 3: Requisição de um recurso específico

5 Conclusão

Com o desenvolvimento deste trabalho concluímos que é possível obter um bom resultado ao implementar comunicação P2P utilizando supernodos e nodos constituindo uma arquitetura híbrida. A troca de informações entre os nodos se mostrou eficaz e ágil, possibilitando-nos aprofundar nossos conhecimentos sobre os tópicos abordados em aula, como P2P e comunicação coletiva. O envio de arquivos executando o código em múltiplas máquinas virtuais funcionou da forma esperada, sendo possível a visualização do recurso que foi requisitado.

Caso tivéssemos entregado a tabela DHT que foi solicitada no escopo do trabalho acreditamos que a comunicação entre os supernodos ao buscar por um arquivo dentro do grupo seria feita de forma mais direta e rápida, tendo em vista que cada supernodo seria responsável por um range de hashes.