Trabalho Prático - 1

Francisco J. Lucca, Marlon Varoni, João Almeida

13 de setembro de 2021

1 Introdução

Neste relatório será descrito a implementação e alguns resultados obtidos sob o trabalho prático 1, estipulado pelo professor Sérgio Johanm. Para este trabalho nos foi solicitado o desenvolvimento de uma aplicação distribuída que modela o acesso concorrente a algum tipo de banco de dados ou arquivo de registro, de tal forma que os processos distribuídos tenham que realizar a sincronização do acesso ao recurso. As operações suportadas são: *insere*, *elimina* e *busca*. As operações possuem pesos diferentes, onde: uma operação de busca pode ser executada inúmeras vezes concorrentemente; uma operação de inserção pode somente ocorrer em paralelo com operações de busca; operações de exclusão não permitem a execução de nenhuma outra operação em paralelo.

Deve existir também um servidor de semáforos que implementará as funções *P()* e *V()* genéricas que serão responsáveis por sincronizar as operações que o servidor irá executar.

2 Organização do código

A organização do código utiliza a arquitetura padrão *RMI* (*Remote Method Invocation*), onde a definição dos serviços remotos é feita através de interfaces, enquanto que o comportamento da aplicação fica em classes que as implementam. A aplicação está dividida entre processos Semáforos, Clientes e Servidores. Mais detalhes sobre a arquitetura estão descritos nas seções abaixo.

2.1 Processo Servidor de Semáforos

Um semáforo é uma variável inteira, compartilhada entre diversos processos. O ganho na sua utilização é a possibilidade de sincronização dos processos e controle de acesso em um determinado recurso comum a todos num ambiente concorrente.

Na classe *SemaphoreServer* instanciamos o semáforo e delimitamos seu valor. No seu método *main()* é criado o registro Java RMI (caso este ainda não exista) e iniciado o servidor de semáforos.

Para realizar a sincronização entre processos utilizam-se os métodos *P()* e *V()*, que possuem as funções de liberação e proibição de acesso, respectivamente. Quando um processo servidor deseja efetuar alguma operação no recurso compartilhado, primeiro é enviado uma solicitação de acesso ao servidor de semáforos, que levará em conta as regras abaixo antes de liberar ou não o acesso:

1. Caso a operação seja de *Busca*, seu peso é 0. Sendo assim, diversas buscas podem ser executadas simultaneamente.

2. Caso a operação seja de *Inserção*, seu peso é 8. Isso significa que as inserções são mutualmente exclusivas entre si, mas ainda possibilitam a execução de buscas ao longo de seu processamento.

3. Caso a operação seja de *Eliminação*, seu peso é 10. Isso significa que as eliminações são mutualmente exclusivas entre si, impedindo qualquer outra operação de ser executada concomitantemente.

A classe *SemaphoreServer()* possui o método *main()* onde são realizadas as conexões RMI com os servidores. Também são implementados os métodos *P()* e *V()* que são responsáveis por realizar uma adaptação

do protocolo baseado nas operação *wait* e *signal*. A solução encontrada foi fazer pequenas modificações no código destes métodos já conhecidos para ser possível o cumprimento das regras de limitação à região de recurso compartilhado citadas no enunciado. Para isso, cada operação solicita um acesso através de *P()* onde é informado um valor como parâmetro. Este valor decrementa uma variável chamada *PermissionValue*, portanto quando outro servidor requisitar acesso, poderá ficar aguardando até que seja liberada a permissão. O acesso é liberado quando é incrementado o *PermissionValue* e chamada a função *notifyAll()* para permitir que outros processos bloqueados por *wait()* sejam iniciados.

2.2 Processos Servidores

Os processos servidores conectam-se ao processo servidor de semáforos através de um endereço IP e uma porta, com a intenção de enviarem suas solicitações de execução de operação. No seu método *main()* é criado o registro Java RMI (caso este ainda não exista) e iniciado o servidor.

O servidor também é responsável pela implementação dos métodos de *Busca()*, *Insere()* e *Deleta()*. Vale ressaltar que o servidor, apos cada execução de método aplica um intervalo de 3 segundos para melhor exemplificar o processo de exclusividade de cada método.

Cada servidor executa em seu construtor uma etapa para a criação de um arquivo *Database.txt*. Os métodos da classe Servidor podem ser chamados por clientes, e quando executados são responsáveis por fazer a conexão e manipulação do arquivo criado, assim como também retornar *logs* informando se a operação foi realizada com sucesso. Quando um método é chamado, ocorre a comunicação com o servidor Semáforo, requisitando permissão de acesso por meio da operação *P()*, caso o acesso for permitido, a execução continua, e após passar pela fase de busca, inserção ou exclusão dos dados, é realizada a operação *V()* para liberar o acesso para outros clientes.

2.3 Processos Clientes

Os processos clientes conectam-se a um dos processos servidores através de IP e porta. Eles fazem as chamadas de *Busca()*, *Insere()* ou *Deleta()* para o processo servidor, passando uma variável do tipo *string* como parâmetro (o que aqui equivale a um item a ser pesquisado, inserido ou deletado do database).

Cada cliente possui um método chamado *runRandomly()*, responsável por fazer a chamada das operações de busca, inserção e exclusão de forma aleatória. Este método recebe do usuário, por meio da linha de comando, a informação da porcentagem de cada operação a ser chamada de forma aleatória. Essas chamadas são feitas a cada dois segundos até que se encerre a execução daquele cliente em específico. Para isso foi utilizado o método *Random()* do Java para gerar um número aleatório de 1 a 100. Após ser retornado o número aleatório, se estiver na faixa de porcentagem definida pelo usuário, a operação é realizada.

Ao executar um processo cliente também é exibido um "Menu de opções"onde é possível escolher por linha de comando qual tipo de operação será realizado ou se serão feitas chamadas aleatórias para o Servidor. No método *main()* da aplicação é feita a conexão via RMI com o Servidor desejado, e também é obtido a informação de qual operação será executada para posteriormente ser chamado o método correspondente da classe Servidor.

3 Instanciando servidores e clientes

Para realizar um teste capaz de validar se o projeto está cumprindo o que foi proposto no enunciado recomenda-se o uso de instâncias de servidor, instâncias de clientes e um único servidor de semáforos. Para compilar o projeto é usado o comando *make* na raiz do mesmo, configurado pelo arquivo *makefile* para compilar todas as classes existentes. Após a compilação ser bem sucedida basta executar os comandos listados abaixo para cada processo:

```
$ java SemaphoreServer <ip> <port>

$ java Server <ip> <port> <SemaphoreServer_ip> <SemaphoreServer_port>

$ java Client <Server_ip> <Server_port> <Client_ip> <nickname>
```

Para funcionamento correto, é importante fazer a execução na ordem citada nos exemplos acima, onde primeiramente será iniciado o servidor de semáforos e posteriormente o servidor e cliente. Especificamente no cliente ainda serão solicitadas informações do usuário como o tipo de operação a ser realizado ou a porcentagem de cada uma. O *nickname* é utilizado para diferenciar os *logs* de cada cliente que será exibido no servidor. Mais detalhes sobre esta etapa estão detalhados na Figura 1.

4 Demonstração da Implementação

Para esta demonstração serão executados:

1. Um processo servidor de semáforo
2. Um processo servidor
3. Dois processos clientes

Na Figura 1 estão descritos por meio dos terminais como são inicializados os processos da aplicação, pela imagem é possível ver as conexões por RMI sendo feitas, e também a criação do arquivo *Database.txt*.

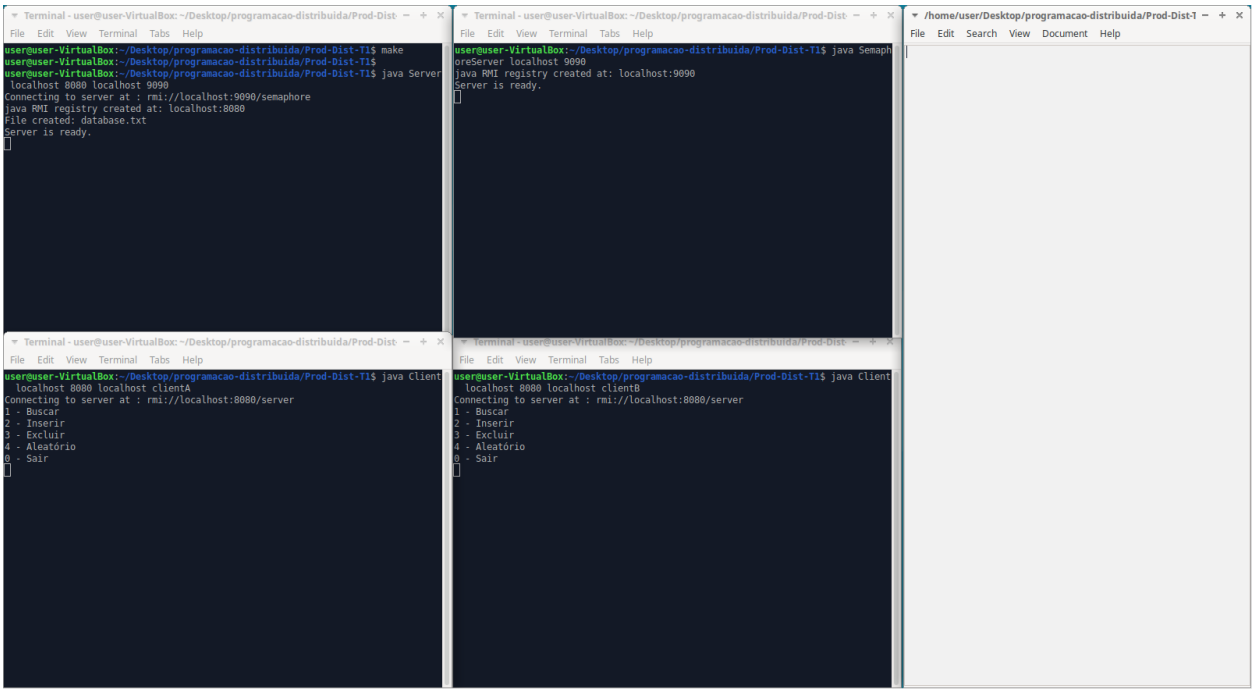

Figura 1: Configuração inicial

Na Figura 1 também é possível observar que o arquivo *Database.txt* está aberto e no momento não há dados armazenados.

Na Figura 2 o cliente A realiza uma inserção e a *string* adicionada é exibida no arquivo.

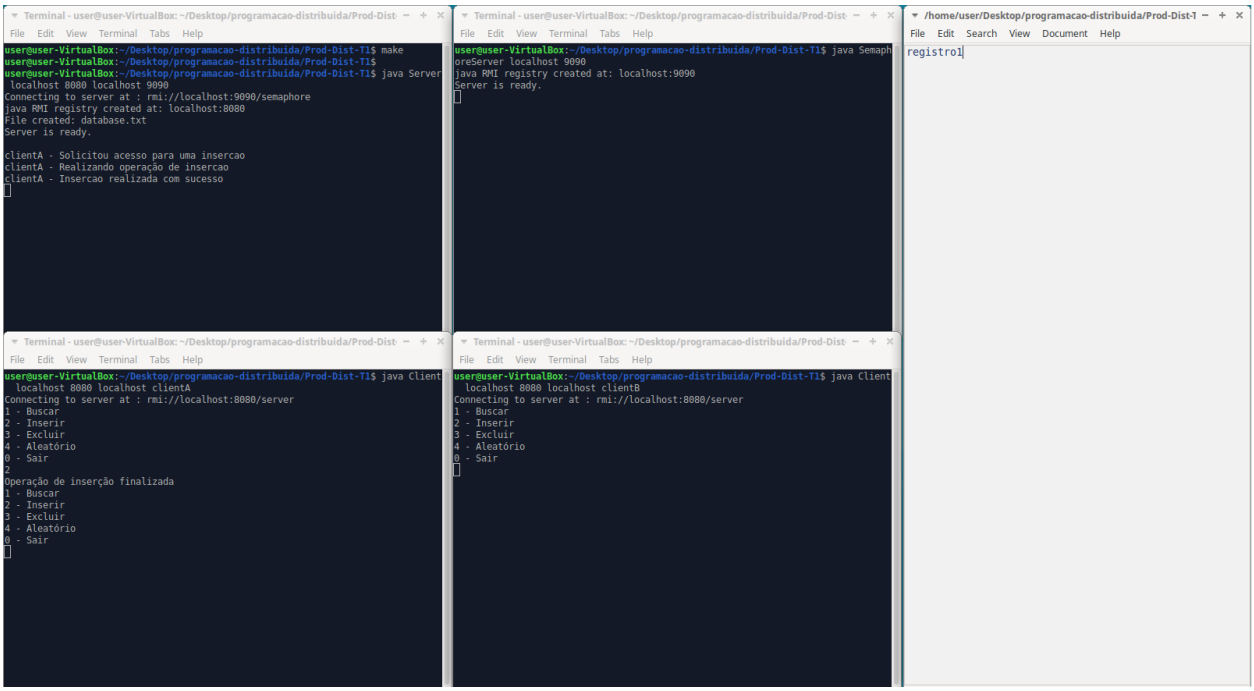

Figura 2: Cliente A insere um registro

Na figura 3 o cliente B solicita a busca de um registro. Neste caso, a busca poderia ser realizada de forma concorrente à uma inserção, por exemplo.

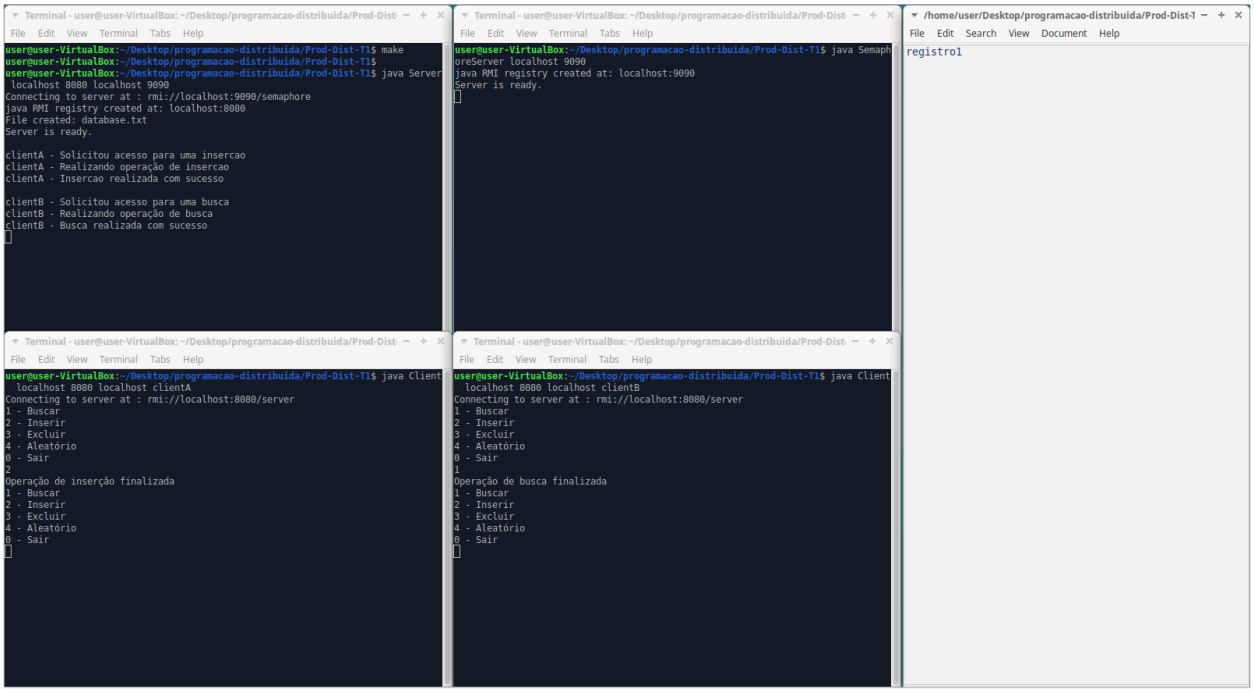

Figura 3: Cliente B busca um registro

Na figura 4 o cliente A exclui um registro que não é mais exibido no arquivo. Caso uma inserção estivesse em andamento, a exclusão não seria feita até que a inserção fosse finalizada.

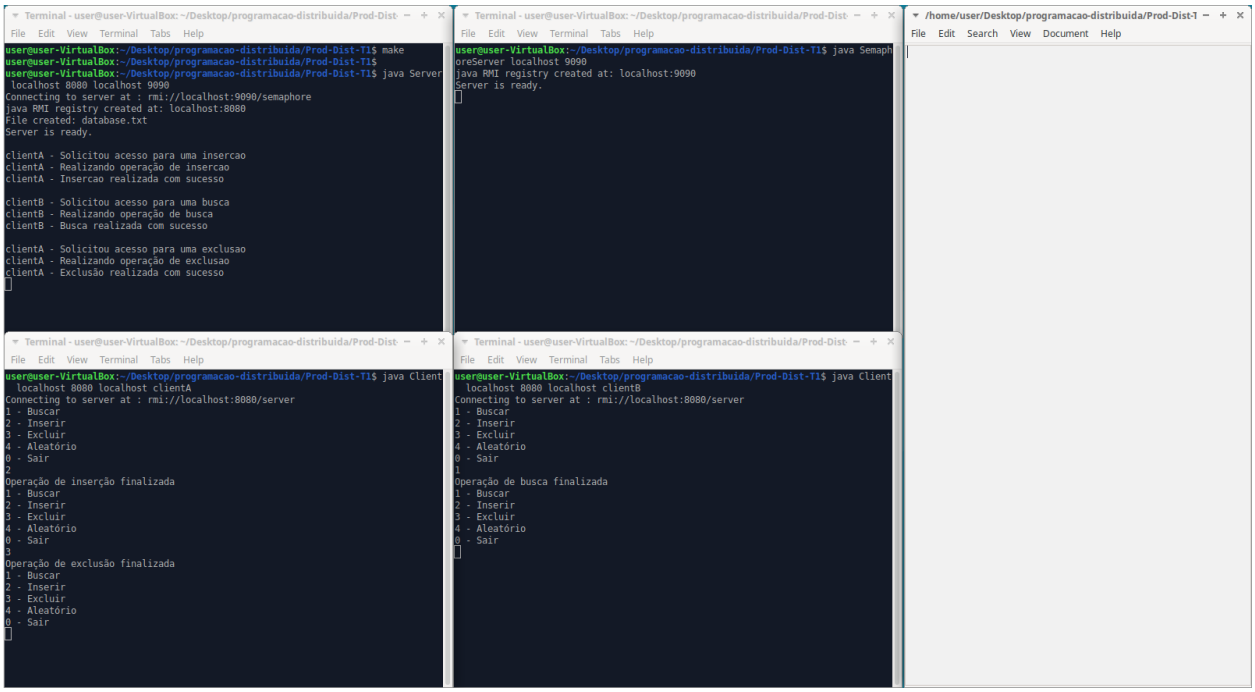

Figura 4: Cliente A exclui um registro

Na figura 5 o cliente A e B realizam uma inserção ao mesmo tempo, o cliente A aguarda até que o cliente B finalize a operação para que seja liberado o acesso.

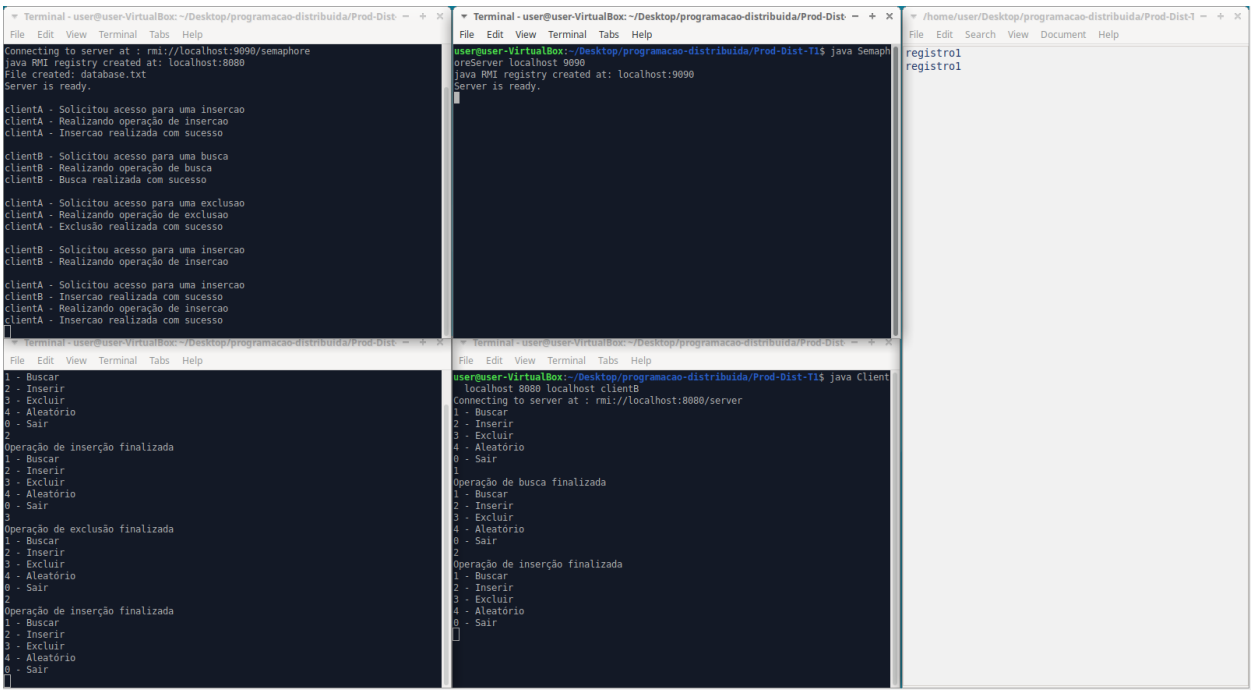

Figura 5: Clientes A e B inserem um registro

Na figura 6 é realizada uma busca e uma inserção ao mesmo tempo.

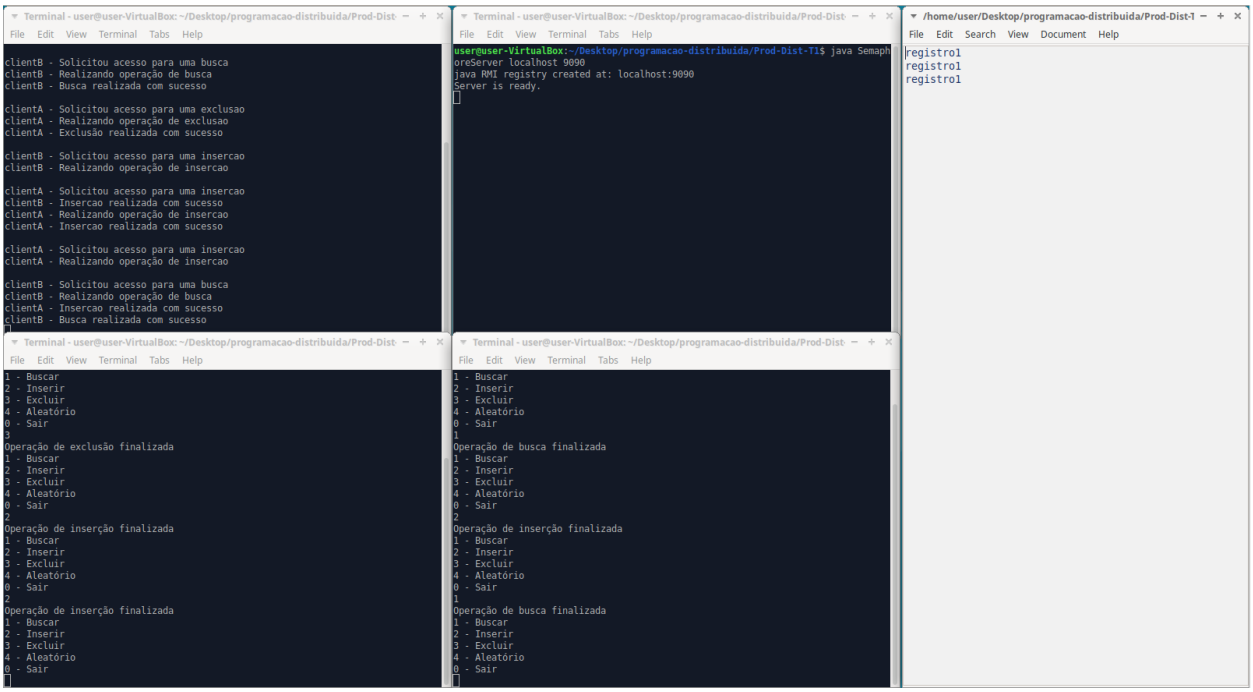

Figura 6: Clientes A insere e B busca um registro

5 Conclusão

Foi concluído pelo grupo com a realização desta atividade que podem ser obtidos resultados satisfatórios com o uso de Java RMI para a invocação de métodos que residem em diferentes máquinas virtuais Java (JVM), possibilitando a conexão entre múltiplos clientes e servidores.

A implementação dos métodos P() e V() viabilizaram, na prática, um melhor entendimento do funcionamento de um servidor de semáforos, também sendo indispensáveis ao possibilitar a troca de mensagens entre os múltiplos servidores da aplicação e o controle de acesso à um recurso compartilhado.