UNIFACS

UC SISTEMAS DISTRIBUÍDOS

RELATÓRIO DO PROJETO A3

Supervisionado pelos professores:

Adailton Cerqueira

Wellington Lacerda

INTEGRANTES:

Bruna Suelen Garcia Rosa Silva - RA 12722110390

Heidielton Carmo de Brito - RA 1272022645

Matheus Filipe Chaves da Cruz - RA 12720110306

Tiago Rocha dos Santos - RA 1272022594

Willian Ribeiro de Araújo - RA 1272019816

Salvador-BA

2023

Sumário

1. INTRODUÇÃO

A motivação do trabalho consiste em criar um aplicativo utilizando sockets que simule a captação de dados de venda de uma rede de lojas, onde os vendedores realizam vendas de produtos e os gerentes consultam as informações sobre as vendas registradas dos vendedores e produtos. Todas as operações devem ser realizadas a partir de trocas de mensagem pela rede de comunicação, a arquitetura escolhida foi a cliente/servidor utilizando sockets.

Sobre o conceito de sockets podemos afirmar que:

> O termo socket se refere a um canal de comunicação entre os protocolos de Internet e as aplicações, sendo identificado pelo protocolo utilizado e pelo endereço local ou remoto. Uma dada aplicação pode acessar diferentes sockets, sendo que um único também pode ser acessado por várias aplicações, Figura 3.1. A comunicação por meios de sockets se dá considerando a estrutura Cliente/Servidor. Neste caso, tanto o cliente como o servidor criam os seus respectivos sockets. No computador servidor, o socket é configurado para verificar continuamente se algum cliente está solicitando uma conexão. Em caso positivo, o servidor aceita a conexão e começa a se comunicar com o cliente. A comunicação só é interrompida quando a conexão é finalizada em um dos dois extremos. (PIRES, 2014, p. 15).

Foi utilizado a linguagem de programação Java, toda comunicação entre os processos foi feita utilizando socket através da rede local e o banco de dados utilizado é o SQlite, na nossa aplicação o servidor é o responsável por receber as vendas dos vendedores e persistir esses valores no banco de dados, além de disponibilizar consultas para os gerentes sobre as informações das vendas gerais ou detalhadas. Os clientes são os processos Gerente e Vendedor, eles conectam-se ao servidor para realizar as suas operações de venda ou busca. É exigido que um cliente assuma a responsabilidade de servidor temporário caso o servidor principal esteja indisponível, para isso é

necessário que haja uma eleição de um líder para assumir a função de servidor temporário.

O algoritmo de Bully é um algoritmo de eleição (ou eleição de líder) utilizado em sistemas distribuídos para selecionar um líder entre vários processos. No algoritmo, cada processo inicia uma eleição quando percebe a falha do líder atual. Os processos com IDs mais altos enviam mensagens de eleição para os processos com IDs mais baixos, e o processo com o maior ID se torna o novo líder. Esse algoritmo garante que o processo com o maior ID seja eleito como líder, garantindo a continuidade do sistema distribuído.

2. REQUISITOS NECESSÁRIOS PARA EXECUÇÃO DA APLICAÇÃO

Para executar a aplicação, serão necessários alguns requisitos para tornar possível a execução. Abaixo os requisitos:

- Java Development Kit (JDK) 8
- Java Runtime Environment (JRE) 8
- IDE caso quiser visualizar e compilar o projeto
- Principais bibliotecas utilizadas:
  java.io.BufferedReader; java.io.IOException;
  java.io.InputStreamReader; java.io.PrintWriter; java.net.Socket;
  java.io.FileInputStream; java.io.FileNotFoundException;
  java.util.Properties;
  java.util.Random;
  java.net.ServerSocket;
  java.io.File;
  java.sql.Connection;
  java.sql.DriverManager;
  java.sql.SQLException;
  java.sql.Statement;
  java.util.logging.Level;
  java.util.logging.Logger;
  java.util.Objects;
  java.sql.PreparedStatement;
  java.sql.ResultSet;
  java.text.ParseException;

3. INSTRUÇÕES PARA EXECUÇÃO DO PROJETO

O repositório está disponível no seguinte link "https://github.com/WillianR381/trabalhoSDA3" presente no Github, caso o usuário tenha o git instalado em seu computador pode optar por fazer o clone do projeto executando o seguinte comando "git clone https://github.com/WillianR381/trabalhoSDA3.git", mas se não tiver o git instalado, o mesmo pode fazer o download do arquivo zip e extraí-lo posteriormente.
Para executar o projeto, criamos dois "scripts" para inicializar a aplicação. Pensando nos sistemas operacionais mais utilizados, criamos scripts para Windows e Linux:

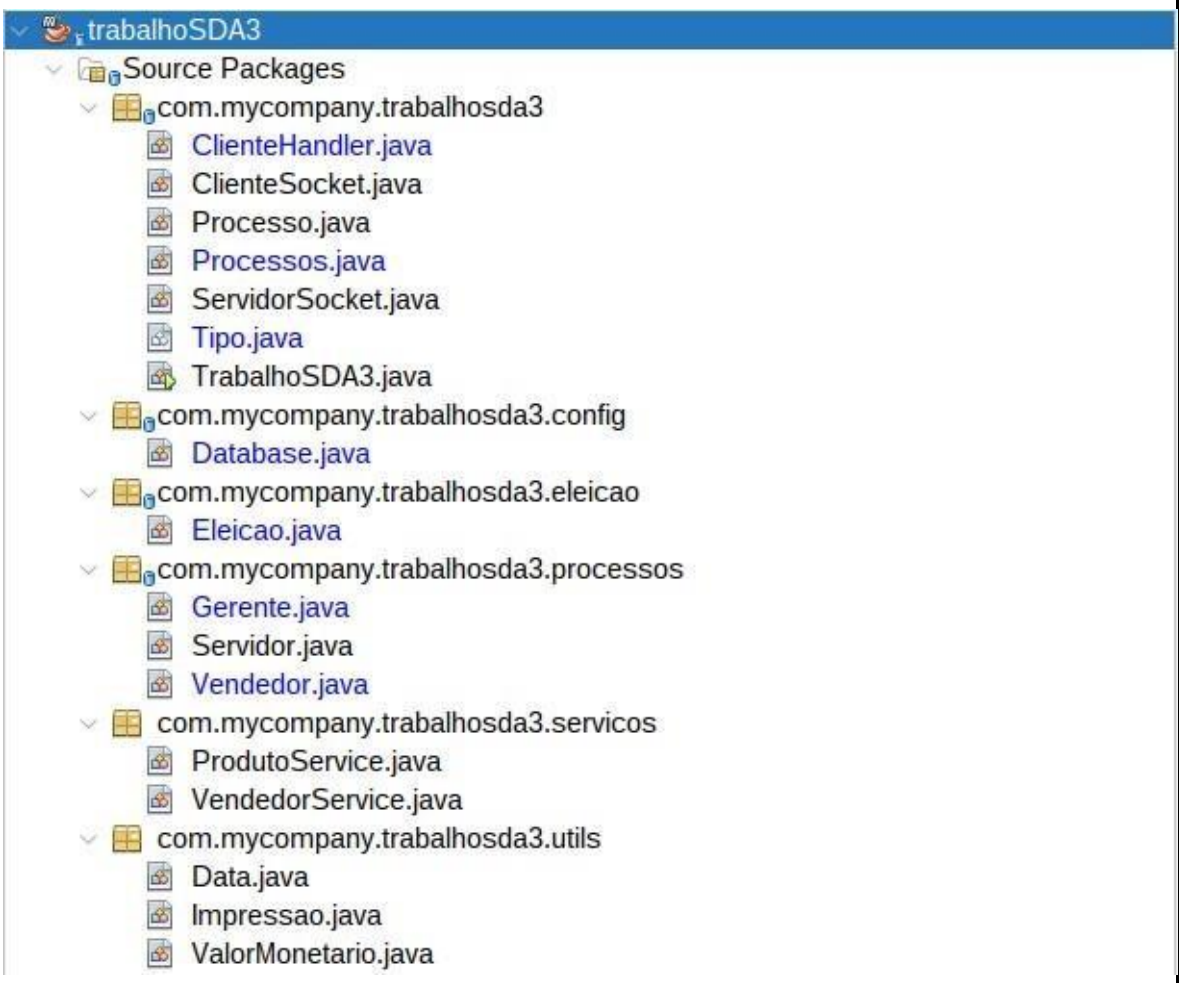
- Windows: Na pasta raíz, contém a pasta "scriptsIniciacaoWindowns". Lá contém os arquivos "iniciaServidor.bat" e "iniciaTodosProcessos.bat". Sugerimos abrir o arquivo "iniciaTodosProcessos.bat" que já inicia todos os processos, inclusive o servidor.

- Linux: Na pasta raíz, contém a pasta "scriptsIniciacaoLinux". Lá contém os arquivos "iniciaServido.shr" e "iniciaTodosProcessos.sh". Sugerimos executar o arquivo "iniciaTodosProcessos.sh" que já inicia todos os processos, inclusive o servidor.

4. DESENVOLVIMENTO

Explicação das Classes A estrutura de pasta e classes do nosso projeto é visível na figura 1. Descreveremos as pastas e classes a seguir:

Figura 1



Lista de arquivos e pastas do projeto

4.1 TRABALHOSDA3

Nessa pasta, contém a nossa classe principal do projeto chamada TrabalhoSDA3.java. Ela é responsável por iniciar os processos, recebe como argumento qual o tipo do processo que será iniciado, ou seja, Gerente, Vendedor ou Servidor, o nome de cada processo, identificador e a porta que utilizará, esses valores são da execução do script ***iniciaTodosProcessos.sh***. Apresenta uma estrutura

condicional que verifica o valor da variável tipo, instanciando sua classe referente para iniciar o processo escolhido. No qual será explicado posteriormente na pasta Processos.

ClienteHandler.java

A classe contém o código que é executado quando um nova thread é iniciada, possibilitando várias requisições simultâneas, a classe manipular as requisições do ***ClientSocket***, a partir da mensagem de operação recebida, ela realiza a ação esperada, podendo ser de “venda”, validando e persistindo os dados das vendas no banco de dados, caso for de “busca”, será enviado ao ***Gerente*** todas as opções de busca disponíveis, recebendo a opção desejada e retornando os valores da busca, caso recebe a mensagem de operação “ping” basicamente retorna ao cliente a string “Ok” para informar que o servidor está funcionando. A classe também possibilita a troca de mensagens entre os processos quando ocorre uma eleição, ao receber uma mensagem de operação “iniciaEleicao” define como True o valor da variável que identifica se está ocorrendo uma eleição ou não, se o método chamado for o ***verificaProcessoComIdentificadorMaior()*** retorna o seu identificador para o cliente que solicitou, o método ***manipulaProcessoComIdentificadorMaior()*** o processo atual assume o papel de eleição e tenta eleger-se líder, isso é feito ao executar o método ***verificaProcessoComIdentificadorMaior()*** da classe de eleição, caso a mensagem for “novoLider” recebe o identificador do líder eleito e atribui à variável líder na classe Processo.java e a última mensagem operação disponível é a “encerraEleicao” atribuindo como False o valor da variável que identifica se está ocorrendo uma eleição.

Processos.java

Classe singleton responsável por gerenciar e mapear todos os processos, possui os atributos processos que é uma lista de processo, o atributo “me” armazena o objeto do Processo com os meus valores, tornando fácil pegar os atributos do meu processo , caso eleito um líder é armazenado o objeto do Processo os valores de host, porta, nome e identificador desse líder, atributo servidor armazena o Processo servidor do projeto. Os Métodos getters e setters, o método ***importaArquivoConfiguracao()*** responsável por pegar o valores dos processos no arquivo de configuração, onde há uma iteração sobre cada item no arquivo de configuração e adicionando na lista de processos uma instância com os valores atual do laço de repetição, o método ***servidorRodando()***, onde verifica se a conexão com o servidor está ativa retornando um booleano dessa condição.

Processo.java

Uma classe com os atributos nome, identificador, host e port e seu getters e setters, tornando mais fácil acessar e manipular os valores de cada processo que está rodando no terminal.

Tipo.java

Tipo é uma classe abstrata que será herdada por outras classes, possui como atributos a porta, nome, identificador, uma instância do ***ServidorSocket*** e a variável que armazena a variável mensagemOperacao. O atributo **serverSocket** é do tipo da classe **ServidorSocket**, além disso a classe Tipo possui um construtor com os seguintes atributos: porta, nome e identificador. Essa classe possui um método chamado ***iniciarConexao()*** que inicia um servidor socket na porta passada no construtor, também apresenta o um método abstrato que será implementado pela classe filha. As classes ***Servidor***, ***Gerente*** e ***Vendedor*** são classes “filhas”, ou seja, estendem da classe ***Tipo*** e serão explicitadas posteriormente.

ClienteSocket.java

Classe responsável por iniciar a conexão com o servidor socket, contém métodos para enviar e receber mensagens entre cliente e o servidor. O construtor da classe recebe como parâmetro o host e a porta do servidor que se conectará e uma socket para possibilitar a conexão ao servidor.

4.2 CONFIG

Nessa pasta apresenta apenas a classe Database.java, no qual foi desenvolvida no padrão singleton, apresenta o método *getInstance* que é responsável por retornar a variável de conexão com o banco de dados, o método ***criaTabelas()*** executa os scripts para criação das tabelas vendedores, produtos e vendas no banco de dados, o método ***insereRegistros()*** é responsável por inserir registros iniciais nas colunas de vendedores, produtos e vendas, assim o banco já vai começar com dados populados, facilitando a visualização das operações do projeto.

4.3 ELEICAO

Nessa pasta apresenta apenas a classe ***Eleicao***, no qual foi desenvolvida no padrão singleton, apresenta atributos e métodos para serem utilizados no processo de eleição de um líder, o algoritmo de eleição escolhido foi o bully, através da sua instância um processo pode chamá-la e conseguir eleger um líder. Cada método presente na classe possui uma variável chamada de mensagemOperacao, ela identifica qual operação será enviada para instância da classe **ClienteHandler** dos processos, assim realizando a ação correspondente. Essa etapa de eleição acontece através de trocas de mensagens entre o processo que iniciou ou está operando a eleição e/os outros

processos, no caso esses processos são o ***Gerente*** e/ou ***Vendedor***. A eleição ocorre a partir das seguintes etapas:

1ª etapa: Início do processo de eleição, ocorre chamando o método ***iniciaEleicao()***, ele é responsável por enviar uma mensagem para todos os processos que uma eleição foi iniciada, essa mensagem é recebida pelos processos e atribuída que uma eleição está ocorrendo, assim impedindo de ocorrer 2 processos eleitorais ao mesmo tempo. Após isso invoca o método ***verificaProcessoComIdentificadorMaior()*** dando início a próxima etapa.

2ª etapa: Etapa responsável por conferir se existe um processo com identificador maior que processo atual, começa enviando uma mensagem requisitando o identificador de todos processos, conforme recebe esses identificadores como resposta, verifica se o identificador do processo é maior que o do processo que veio como resposta, caso não, ele continua essa verificação até iterar sobre do laço de repetição dos processos, não encontrando um identificador maior que o processo atual, seguirá para as 3ª etapa e 4ª etapa. Caso encontre um processo com identificador maior que o processo atual é invocado o método ***manipulaProcessoComIdentificadorMaior()***, responsável por enviar ao processo com identificador atual, encaminha a esse processo uma mensagem de operação informando que esse processo vai assumir a execução da eleição e efetuando a 2ª etapa do processo de eleição. De maneira geral, a 2ª etapa ocorre em recursão (chamando a si mesmo) até encontrar o processo com maior identificador e encaminhando para as próximas etapas do processo eleitoral (3ª etapa e 4ª etapa) .

3ª etapa: Ocorre quando o método ***notificaNovoLider()*** é invocado, encaminha para todos os processos a mensagem de operação mais o identificador do líder eleito, esses processos recebem esse identificador e atribuem pelo identificador do processo como líder.

4ª etapa: Método ***encerraEleicao()*** é responsável por notificar que a eleição foi encerrada para todos os processos, eles recebem e atribui a variável que controla se há ou não uma eleição ocorrendo com o valor false.

O algoritmo apresenta a prevenção de falha, caso um processo não obtiver sucesso ao tentar se conectar com o servidor temporário, ele iniciará uma nova eleição para eleger um novo líder para ser consumido como servidor temporário por outros processos

4.4 PROCESSOS

Essa pasta apresenta as classes responsáveis por iniciar os processos do projeto, cada processo desempenha o papel no sistema, eles são o ***Gerente***, ***Vendedor*** ou ***Servidor***, todas essas classes da pasta fazem herança da classe ***Tipo***.

GERENTE

A classe Gerente estende da classe ***Tipo***, possui um método construtor com os atributos porta, nome e identificador. Ela dispõe do método ***run(),*** que contém também a lógica executará para fazer a busca das vendas no servidor, possui uma estrutura de controle, no primeiro momento verifica se o servidor está respondendo, essa condição sendo verdadeira, utiliza o host e a porta do servidor e instancia a classe ***ClienteSocket*** para fazer a conexão com o servidor através do sockets, permitindo a troca de mensagens entre o cliente-servidor, assim possibilitando esse processo realizar a buscas sobre a vendas no servidor. O servidor não respondendo, o processo verifica se já existe um líder eleito, sendo verdadeiro o processo atual utilizará o host e a porta do líder, assim o líder assumindo o papel de servidor temporário que age igual ao servidor principal, caso não exista um líder o processo atual uma eleição, descrita na classe ***Eleicao***. Apenas um processo pode iniciar e/ou assumir o processo eleitoral. A classe também executa o método ***iniciaConexao(),*** no qual inicia um servidor socket, possibilitando receber mensagens de outros processos, como as mensagens de operação presentes na eleição, além de iniciar o servidor temporário que pode ser consumido pelos processos***.***

VENDEDOR

Conforme a classe Gerente, citada anteriormente, a classe Vendedor é uma classe "filha" de Tipo, e implementa o método ***run()*** contém a lógica executará para fazer o cadastros das vendas no servidor, O método ***iniciarConexao()*** possui o mesmo comportamento citado na classe anterior ***Gerente***.

SERVIDOR

A classe Servidor é "filha" da classe Tipo, possui o mesmo construtor da classe pai e o método ***run()*** que possui dentro o ***iniciarConexao()***. É o servidor principal do projeto, possibilitando aos outros processos consultas de vendas por parte do processo Gerente e cadastrar uma venda a pedido do processo Vendedor.

4.5 SERVICO

Na pasta SERVICOS disponibiliza as classes responsáveis por implementar as operações de busca e/ou cadastro dos processos ***Produto*** e ***Vendedor*** no banco de dados. Cada método presente na classe possui um script sql é executa e retorna a busca desejada ou um status da operação, esses scripts podem utilizar valores passados no parâmetro de cada método como variável nas operações de select ou insert do banco de dados, as classes são:

A classe ***ProdutoService*** o método ***totalVenda()*** recebe por parâmetro o identificador do produto, estamos utilizando o nome do produto como identificador, a partir dele é realizado a busca e retornando do método o somatório das vendas do produto desejado. O método ***totalVendasPorPeriodo()*** recebe no parâmetro a data inicial e final que deseja pesquisar e retornando os nomes do produto e o total de vendas nesse intervalo de datas pesquisado. O método ***melhorProduto()*** retorna o produto que foi mais vendido .

A classe ***VendedorService*** o método ***realizaVenda()*** é responsável por cadastrar uma venda no banco de dados, recebe no parâmetro o nome do vendedor, nome do produto vendido, a data da venda e o total vendido, e retornando do método uma string de status, confirmando ou não que o cadastro ocorreu com sucesso. O método ***totalVenda()*** recebe por parâmetro o identificador do vendedor, estamos utilizando o nome do vendedor como identificador, a partir deste identificador é realizado a busca e retornando do método o somatório das vendas que o vendedor realizou. O método ***melhorVendedor()*** retorna o vendedor que mais vendeu .

4.6 UTILS

Dentro da pasta utils está presente classes que são responsáveis por formatar ou utilitária para o projeto:

A classe ***Data*** é responsável por formatar as datas de entrada e saída. Servindo principalmente para formatar a data, geralmente é utilizada nas operações no banco de dados, possuindo dois métodos, um chamado ***formataParaDiaMesAno()***, que recebe um input na forma de string e transforma para o tipo data, e tem como função a transformação de ano/mês/dia para dia/mês/ano e o outro chamado ***formataParaAnoMesDia()***, que é responsável por alterar uma string para o tipo data e transformá-la de dia/mês/ano para ano/mês/dia. Método ***validaData()*** é responsável por verificar se a String que recebe o valor de data é válida, retornando o booleano da condição.

A classe ***Impressao*** possui um método chamado ***noTerminal()*** que recebe como parâmetro uma string e formata e imprime no terminal. Tivermos muita dificuldade em imprimir uma mensagem vinda de uma comunicação de sockets, onde não conseguimos imprimir toda a mensagem do socket caso estivesse uma quebra de linha, para solucionar isso tivermos que realizar uma adaptação, caso na string estiver “<novaLinha>” é realizado a impressão da string com a quebra de linha.

A classe ***ValorMonetario*** contém somente um método chamado ***formatar()*** que é responsável por receber um valor do tipo double e retornar o símbolo do real (R$) , junto com o valor com obrigatoriamente 02 (duas) casas decimais.

5. Por que escolhemos o Algoritmo de Bully?

A escolha do algoritmo de Bully em relação a outros algoritmos de eleição pode depender das necessidades e características específicas do sistema distribuído em questão. No entanto, existem algumas razões pelas quais escolhemos o algoritmo de Bully :

1-Simplicidade: O algoritmo de Bully é relativamente simples de ser entendido e implementado, o que pode ser vantajoso em sistemas distribuídos com requisitos menos complexos.

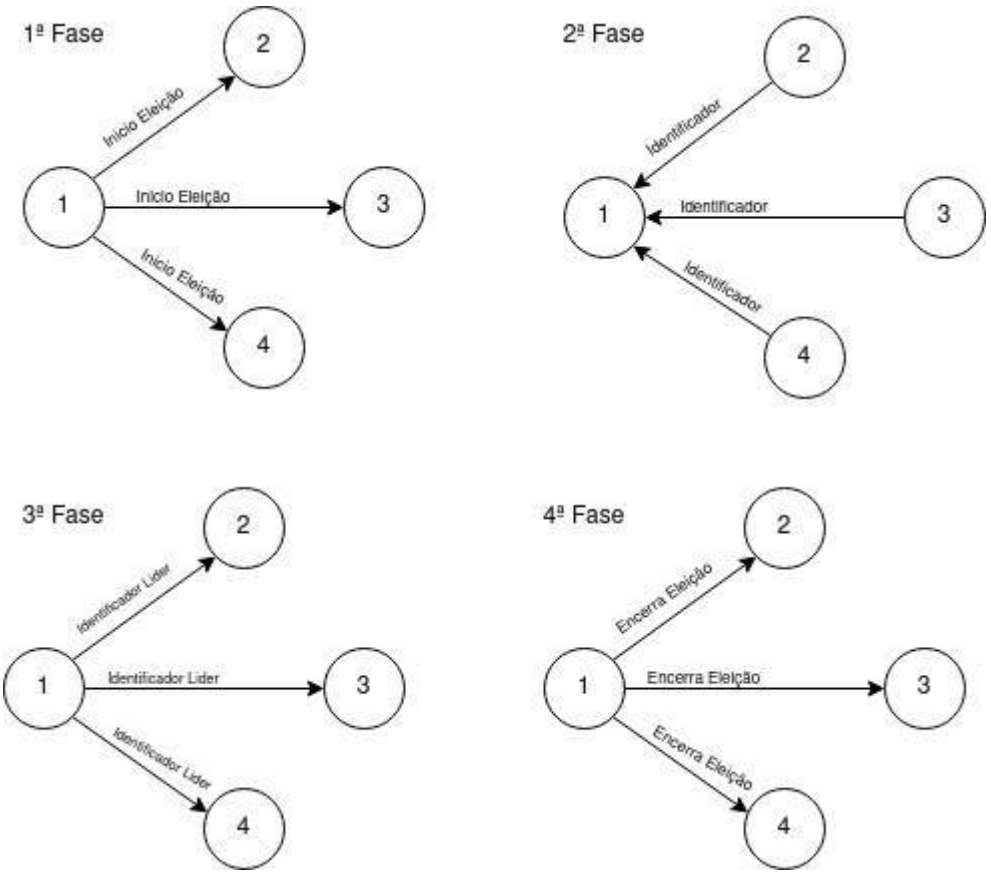
2-Robustez: O algoritmo de Bully é projetado para lidar com falhas de liderança de forma eficiente. Quando um líder falha, o algoritmo inicia uma nova eleição e seleciona um novo líder rapidamente.

3-Escalabilidade: O algoritmo de Bully pode ser escalado para sistemas com um grande número de processos, uma vez que a eleição é realizada apenas entre os processos que estão ativos naquele momento.

No entanto, consideramos outros fatores importantes para nossa escolha, como o desempenho, a latência da rede, a tolerância a falhas e as necessidades específicas do sistema distribuído.

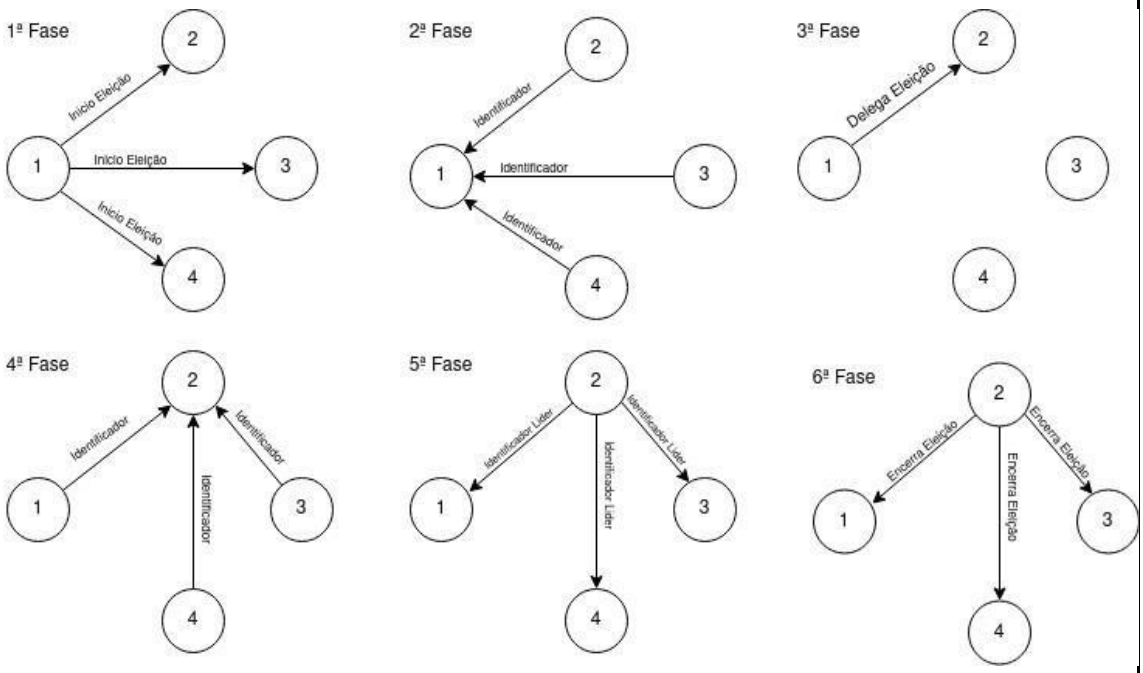
Explicação da implementação do nosso algoritmo de eleição do projeto através dos sockets

Figura 2 - Representação visual do processo elege-se líder pois não encontra um identificador maior que o seu.



Fonte: Elaboração própria

Figura 3 - Representação visual do processo que encontra um identificador maior que o seu



Fonte: Elaboração própria

Ilustração do funcionamento do nosso algoritmo de eleição bully, considerando 4 processos nomeados de 1 a 4, podendo ser visualizado na figura 2 O processo 1 envia uma mensagem ao servidor e não obtêm uma resposta, assim detectando que a comunicação com servidor falhou, iniciando o processo de eleição, encaminhando uma mensagem para todos os processos que uma eleição foi iniciada (fase 1 da figura 2), após isso o processo atual tenta elege-se líder, como foi exigido que a eleição ocorresse através de trocas de mensagens entre os processos, é necessário requisitar os identificadores de cada processo (visível na fase 2 da figura 2), assim verificando se existe algum identificador maior que o do processo atual, caso não haja, o processo 1 assume o papel de líder, notificando todos os processos que ele é o novo líder, enviando uma mensagem que um líder foi eleito mais o seu identificador

(fase 3 da figura 2), logo após envia para todos os processos que a eleição foi encerrada (fase 4 da figura 2). Caso o processo 1 encontre um processo que tenha um identificador maior que o seu, ele delega o papel eleitoral para esse outro processo(visível na figura 3 fase 3), no exemplo o processo 2 assume a eleição e tenta elege-se líder, consequentemente repetindo a etapa de receber os identificadores (visível na figura 3 fase 4) , verificar se existe identificador maior, delegar papel de eleição até encontrar o processo com maior identificador, enviando todos os processos que ele é o novo líder junto com o seu identificador (visualizado na fase 5 da figura 3) e também notifica para todos os processos que a eleição foi encerrada (fase 6 da figura 3).

6. CONCLUSÃO

Destacamos que o projeto possibilitou aprofundamento nos temas abordados no semestre, como a aplicação práticas dos assuntos abordados nas aulas, unindo conhecimentos da linguagem de programação Java, manipulação do banco de dados, implementação da arquitetura cliente-servidor através dos sockets e a troca de mensagens entre processos a partir da rede de comunicação. É importante ressaltar também o processo eleição, que é uma reação a uma detecção de falha de um serviço ou aplicação, possibilitando que o atendimento às requisições não seja interrompido quando ocorrer uma falha.

7. REFERÊNCIAS BIBLIOGRÁFICAS

PIRES, Felipe Augusto. Avaliação de métodos de telereabilitação robótica utilizando comunicação TCP/IP e Unity. 2014. Dissertação (Mestrado em Dinâmica das Máquinas e Sistemas) - Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2014. doi:10.11606/D.18.2014.tde-19032015-163939. Acesso em: 2023-06-19.

Paulo Henrique Oliveira. Sockets: o que é e como eles funcionam?. 31/05/2022 .Sockets: o que é e como eles funcionam? (linuxsolutions.com.br).

Anon. Uma introdução a TCP, UDP e Sockets. ([s.d.]). Recuperado 21 de junho de 2023, de https://www.treinaweb.com.br/blog/uma-introducao-a-tcp-udp-e-sockets

Eleição de líder. (2021). Em Wikipédia, a enciclopédia livre. https://pt.wikipedia.org/w/index.php?title=Elei%C3%A7%C3%A3o_de_l%C3%ADder&oldid=61415123

Jucele França de Alencar , Algoritmos para Eleição de Líder em Sistemas Distribuídos (1998) https://repositorio.unicamp.br/Busca/Download?codigoArquivo=489138

Markus Endler, Algoritmos de Eleição(2000) https://www-di.inf.pucrio.br/~endler/courses/DA/transp/Election.pdf

Livro do Colouris/Dollimore/Kindberg. (2013)