PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL

ESCOLA POLITÉCNICA

Bernardo de Cesaro, Bianca Machado, Mathias Gatti e Mayara Oliveira

Trabalho 3 de Programação Distribuída

Professor: Sérgio Johann

Porto Alegre

2020

1. **Descrição do trabalho:**

Objetivo da tarefa é o desenvolvimento de um sistema distribuído que implemente o relógio de Lamport para a ordenação de eventos com ordem total, ou seja, neste caso o relógio considera o índice do processo para formar a linha do tempo lógica de execução.

Como requisitos deste sistema temos que:

- Deve receber um arquivo de configuração (contendo uma lista formada de *id host port chance*);
- Deve receber o número da linha que corresponde ao respectivo *host*.
- Cada *host* pode realizar eventos locais ou enviar mensagens em utilizando datagramas;
- O envio de mensagens deve ser aleatório para um dos nodos do arquivo, que não seja ele mesmo;
- Os eventos são gerados em intervalos de 0.5 a 1 segundo;
- Cada nodo só pode executar 100 desses eventos e parar sua execução ao atingir este número;
- Em caso de envio de mensagem para nodo que **não está em execução**, o erro gerado deve ser tratado e interromper a execução do processo que tentou enviar a mensagem;
- A sincronização entre os *hosts* deve ser feita utilizando relógio de Lamport.

2. **Organização do código:**

Organizamos o código de acordo com a arquitetura usual de API utilizando NodeJS e o *framework* Express. A maior parte da implementação está no arquivo ./src/*app.js*. Veja a estrutura de pastas do projeto na imagem a seguir.

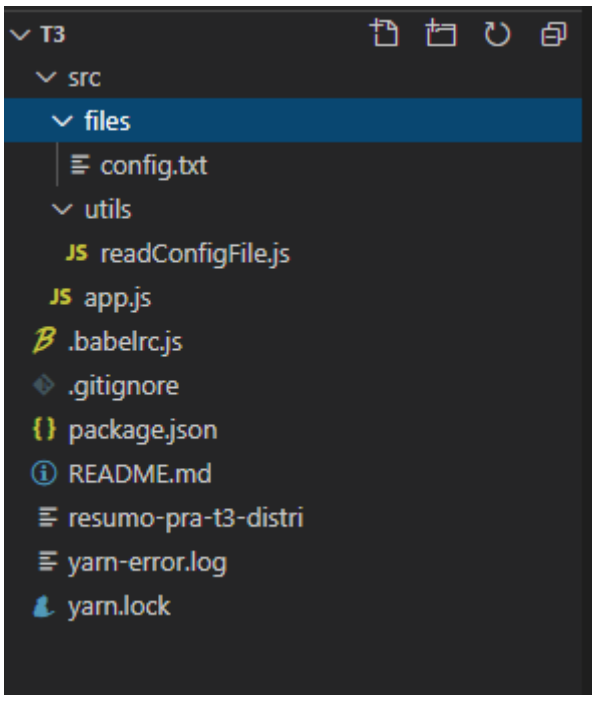

Imagem 1. Organização de pastas do projeto.

Para realizar este trabalho implementamos quatro funções principais *showOptions, ping, startProcess e socket_server.on,* estas são descritas a seguir.

- *showOptions:*
    - Função criada para realizar a configuração inicial do projeto, onde definimos o *id* identificador do processo, o *host* responsável pela execução do processo, a porta onde o processo será executado e a probabilidade da ocorrência de envio de mensagens. Essa função lê a informação do processo escolhido pelo usuário (default ou específico). Com isso, a aplicação já possui a informação do arquivo que deve ler e do processo que precisa encontrar. Após percorrer o arquivo ele salva os dados do processo solicitado, define a porta que o *socket* irá usar e informa os dados que foram armazenados. Ao final, a lista contendo todos os *hosts* encontrados no arquivo é apresentada ao usuário. Caso ocorra algum problema, uma mensagem de erro de leitura de arquivo será enviada.
- *ping:*
    - Algoritmo criado para controlarmos quando os *hosts* estavam ativos ou não. Criamos duas listas, uma para guardar todos os hosts do arquivo *config.txt* e outro para registrar os *hosts* ativos. Basicamente utilizamos um *setInterval* que funciona como um *loop* enquanto todos os nodos ainda não foram ativados, exibimos uma mensagem avisando sobre isto. Após a inicialização de todos os nodos, finalizamos o *setInterval* e continuamos os outros passos do trabalho.
- *startProcess:*
    - É aqui onde iremos colocar os nodos para funcionar junto com o algoritmo de Lamport, quando todos estiverem prontos, este método é chamado periodicamente enquanto ainda houver eventos para serem realizados. Para sermos mais específicos, há a existência de uma variável randômica que irá indicar, conforme as chances do nodo, se o evento que irá ocorrer será um evento local ou um evento externo (ex. Envio de mensagem). Caso seja um evento local, será somado +1 ao contador do relógio local. Caso o evento seja externo, decidimos aleatoriamente um nodo que irá receber uma mensagem, incrementa o contador do relógio e então aplica o a função *sendMessage()* com o valor do relógio lógico.
    - Todos esses possíveis eventos só podem ocorrer até que o nodo realize um máximo de 100 eventos. Chegando neste limite, ele para a sua execução.
- *socket_server.on:*
    - É nesta função que o programa fica escutando mensagens recebidas na porta especificada, mapeamos dois tipos de mensagens: notificação de chegada de um *host* e mensagem de sincronização de relógios. Enquanto as notificações de chegada de *host* estão chegando, esta função exibe a mensagem "*Aguardando os coleguinhas...*" e incrementa o objeto de *availableHosts*. Assim que todos chegam, setamos a variável *started* como *true* e começamos a tratar as mensagens de sincronização de relógios recebidas, que funciona da seguinte forma: incrementa o relógio local com max(valor_recebido, valor_local) + 1, concatena o *id* do host no relógio lógico e exibe mensagem no seguinte formato: *m i relógio rem relogioR*, que significam respectivamente tempo em milissegundos da máquina atual, relógio lógico local, *id* do remetente da mensagem e relógio local do remetente.

3. **Utilização do programa:**

   Para executar o projeto, é necessário ter uma configuração básica:

- NodeJS, em versões recentes (utilizamos v12.19 durante o desenvolvimento e nas instâncias AWS).
- Gerenciador de pacotes *yarn* instalado;

Consulte o *readme.md* disponível no projeto em caso de dúvidas sobre como instalar. Para executar consulte o exemplo a seguir, também disponível no *readme.md.*

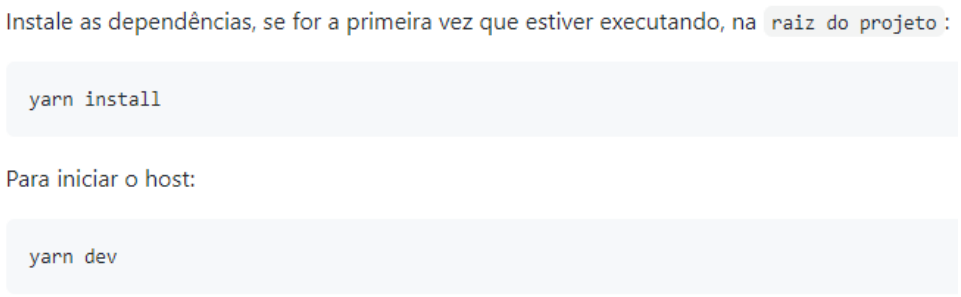

Imagem 2. Dicas para executar o projeto.

4. **Demonstração da implementação:**

A seguir estão as imagens dos testes que realizamos com instâncias EC2 da AWS.

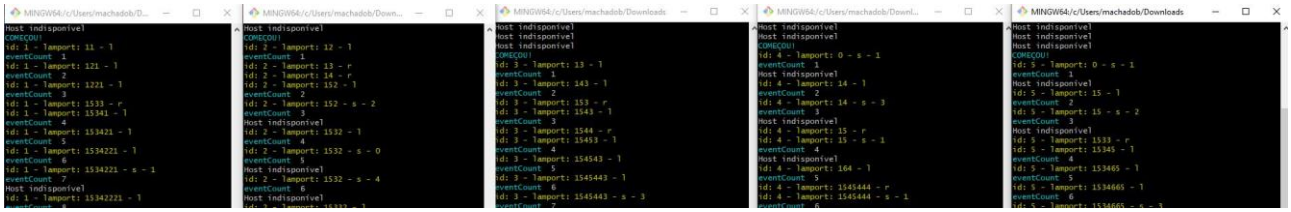

Imagem 3. Início da execução do programa em cinco instâncias EC2.

Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS
Running	t2.micro	2/2 checks ...	No alarms	us-east-1e	ec2-100-26-168-95.compute-1.amazonaws.com
Running	t2.micro	2/2 checks ...	No alarms	us-east-1d	ec2-3-84-162-91.compute-1.amazonaws.com
Running	t2.micro	2/2 checks ...	No alarms	us-east-1a	ec2-3-93-194-248.compute-1.amazonaws.com
Running	t2.micro	2/2 checks ...	No alarms	us-east-1a	ec2-3-88-106-168.compute-1.amazonaws.com
Running	t2.micro	2/2 checks ...	No alarms	us-east-1a	ec2-54-221-13-251.compute-1.amazonaws.com

Imagem 4. Instâncias EC2 da AWS utilizadas.

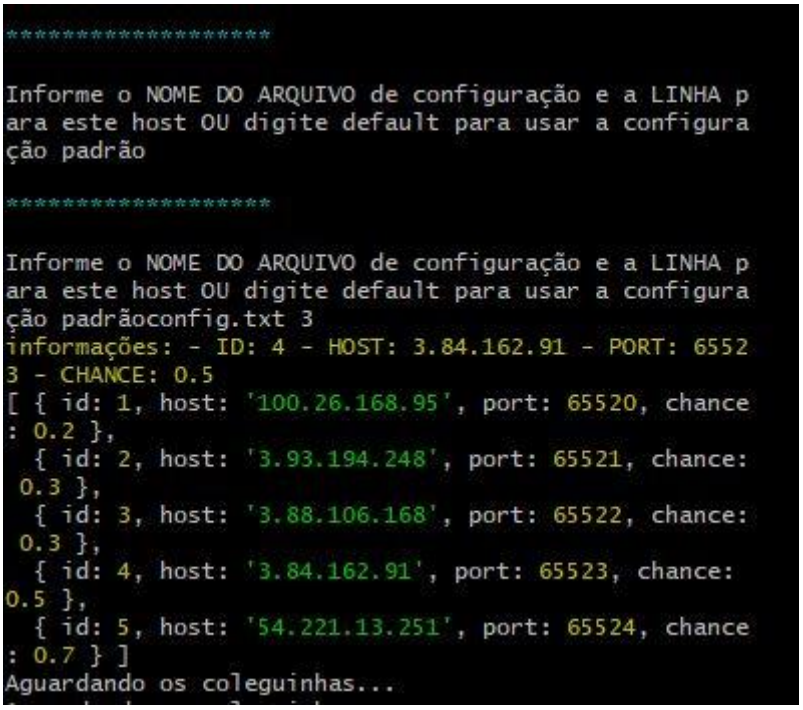

Imagem 5. Hosts do *config.txt*.