# MC714 Sistemas distribuídos
# Relatório do projeto 2
# Algoritmos distribuídos

Daniel de Sousa Cipriano RA: 233228

Novembro 2023

## 1 Introdução

O objetivo do projeto foi utilizar um método de comunicação em sistemas distribuídos (RPC, troca de mensagens com MPI, MQTT, etc), para implementar três tipos de algoritmos comuns de sistemas distribuídos:

- Relógio lógico de Lamport.
- Um algoritmo de exclusão mútua.
- Um algoritmo de eleição de líder ou de consenso distribuído.

## 2 Descrição dos algoritmos escolhidos

### 2.1 Relógio lógico de Lamport

O relógio lógico de Lamport é um algoritmo de sincronização de relógios, que segue o paradigma de ajuste de tempo dos relógios lógicos, não necessariamente pela hora real, mas sim, pela organização dos eventos executados por cada nó na rede. Os relógios lógicos de Lamport definem a relação "acontece antes", na qual, se a e b são eventos de um mesmo processo, então:

- se a acontece antes de b, representamos com: a $\rightarrow$ b
- sendo a o envio da mensagem e, b o recebimento da mensagem, então: a $\rightarrow$ b
- se a $\rightarrow$ b e b $\rightarrow$ c, então a $\rightarrow$ c
- se x e y acontecem em processos diferentes e tanto x $\rightarrow$ y, quanto y $\rightarrow$ x são falsas, os processos x e y são ditos concorrentes, ou seja, não interagem entre si.

O algoritmo de relógio lógico de Lamport segue os seguintes passos:

- É definido um contador de eventos para cada processo no sistema, que será o relógio lógico. Inicialmente o relógio lógico tem valor 0.
- O relógio lógico é incrementado toda vez que um processo realiza um evento.
- Quando ocorre um evento de comunicação entre processos, o evento de envio carrega uma mensagem, contendo o relógio lógico atual do processo remetente.
- No processo destinatário, seu relógio lógico é atualizado para o valor máximo entre seu contador e o contador carregado na mensagem, incrementando 1 ao resultado final: max(contador local, contador carregado em mensagem) + 1.
- Dessa forma, é garantido que o contador sempre irá obedecer as premissas do relógio lógico de Lamport, ou seja, o marcador de hora do destinatário deve ser maior que o marcador carregado na mensagem pelo remetente.

Relógios lógicos de Lamport são utilizados para garantir a ordenação total de eventos, em situações nas quais a execução de atualizações no sistema, devem seguir sempre a mesma ordem, para evitar inconsistências no funcionamento de aplicações.

## 2.2 Algoritmo de exclusão mútua: Token Ring

Algoritmos de exclusão mútua são vitais em sistemas distribuídos, pois garantem a coordenação e sincronização dos nós ao acessar recursos distribuídos no sistema. O algoritmo Token Ring, parte da utilização de uma ficha(token), que é repassada entre os nós da rede, em formato circular. Quando um nó está de posse da ficha, ele pode realizar acesso ao recurso distribuído, sem interferência de outros nós. Ao encerrar a utilização do recurso ou, se o nó não possui interesse em acessar o recurso, a ficha é repassada para o nó vizinho na rede. O algoritmo Token Ring evita a ocorrência de impasses e inanição entre processos, mas há a desvantagem de dificuldade em recuperar uma ficha e refazer o ciclo de comunicação, caso algum dos nós na rede fique inativo.

## 2.3 Algoritmo de eleição de líder: Algoritmo do anel

Em sistemas distribuídos, muitas vezes é necessário definir um nó para desempenhar o papel de coordenador, tomando decisões acerca de permissões de acesso, requisitadas por outros nós. O Algoritmo do anel, segue o princípio de comunicação circular. Um dos nós na rede inicia a eleição de líder, por detecção de inatividade do líder por exemplo, montando uma mensagem com seu identificador de nó. Essa mensagem é repassada para os nós vizinhos na rede, cada nó que recebe a mensagem, acrescenta seu identificador a uma fila de identificadores. Ao fim, quando a mensagem retorna ao nó de origem, ele verifica qual o maior identificador da fila e repassa uma mensagem de coordenador, informando aos outros nós qual o identificador do novo líder eleito.

# 3 Detalhes da implementação

## 3.1 Ambiente de simulação

O ambiente de simulação utilizado na implementação do projeto, foi um ambiente de contêineres Docker. Por padrão, contêineres Docker tem uma sub-rede local atribuída no formato bridge network, compartilhando a rede local com o host da máquina. Isso permite que os contêineres em uma mesma rede se comuniquem entre si, podendo simular execuções comuns em um ambiente de sistema distribuído. No caso da solução desenvolvida, foram criados três contêineres Docker, nomeados como `node1`, `node2` e `node3`.

Os três contêineres foram baseados em uma imagem Python, rodando no sistema Linux Alpine. Foi atribuída uma rede em modo bridge network, personalizada para os contêineres. Os IPs foram definidos como estáticos, variando na faixa de sub-rede `172.18.0.0/16`. O contêiner `node1` teve seu IP definido como `172.18.0.3`, o contêiner `node2` teve seu IP definido como `172.18.0.4` e o contêiner `node3` teve seu IP definido como `172.18.0.5`.

Também foi atribuído um diretório compartilhado entre os contêineres, nomeado como `/resources`, que foi útil na simulação do acesso a recursos compartilhados, no teste do algoritmo de exclusão mútua. Cada contêiner possui um diretório que contém os arquivos de execução de cada tarefa, inicializados no diretório raiz do ambiente, o diretório `/lamport` armazena os arquivos relacionados à tarefa Relógio lógico de Lamport, o diretório /mutex armazena os arquivos relacionados à tarefa Exclusão mútua: Token Ring e o diretório `/leader_election`, armazena os arquivos relacionados à tarefa Eleição de líder: Algoritmo do anel. Durante a execução, é gerado o arquivo `shared_file.txt` em `/resources`, que foi utilizado como teste para o algoritmo de exclusão mútua.

# 4 Linguagem de programação e método de comunicação

A linguagem de programação utilizada na implementação do projeto, foi a linguagem Python. O método de comunicação utilizado, foi troca de mensagens via socket em Python. A biblioteca socket do Python, fornece métodos que permitem realizar conexão e troca de mensagens entre processos, na mesma máquina ou em máquinas diferentes. Sockets utilizam o paradigma cliente/servidor, com cada máquina participante da troca, adotando um socket. As principais primitivas de interface fornecidas pela biblioteca socket são:

- socket: cria um terminal de comunicação
- bind: vincula endereço ao socket
- listen: coloca socket disponível para receber conexões como servidor
- accept: bloqueia o chamador, até que uma conexão seja realizada
- connect: permite estabelecer uma conexão em servidor como cliente
- send: permite envio de dados
- receive: permite recebimento de dados

- close: encerra conexão

Foi utilizada também a biblioteca threading do Python, devido a necessidade de realizar conexões como nós que atuam como cliente e servidor e, como alguns métodos de socket em Python são bloqueantes, foi necessária a utilização de threads de cliente e servidor, para garantir a execução paralela.

Também foram utilizadas as bibliotecas os, json, time e random, para importação de variáveis de ambiente, serialização e desserialização de dados, definir atrasos e simular execuções de processos. Cada diretório em contêiner Docker, contém scripts Python que representam os nós da rede para cada algoritmo. O contêiner `node1` possui os scripts `node1.py`, o contêiner `node2` possui os scripts `node2.py` e o contêiner `node3` possui os scripts `node3.py`.

# 5 Implementação dos algoritmos

Para implementação dos três algoritmos, foi utilizada uma lógica semelhante, com objetos em Python. Todos scripts possuem a classe Process, que representa o processo e seus atributos, para execução dos diferentes algoritmos.

## 5.1 Relógio Lógico de Lamport

O algoritmo de relógio lógico de Lamport, foi implementado através da comunicação entre os nós dos contêineres, `node1` e `node2`. Foi elaborada uma troca bidirecional de mensagens através de sockets de cliente e servidor, distribuídas em threads diferentes. O processo remetente, envia uma mensagem contendo suas informações de processo e seu relógio lógico. O processo destinatário, recebe a mensagem, atualiza seu relógio lógico e imprime a saída com os relógios lógicos antigos de remetente/destinatário e os relógios lógicos atualizados de remetente/destinatário.

## 5.2 Exclusão mútua: Token Ring

O algoritmo Token Ring, foi implementado utilizando os nós dos três contêineres Docker. Foram utilizadas threads de cliente e servidor, para estabelecer a comunicação sequencial e circular entre os processos na rede. O processo remetente, envia mensagem com seu identificador e o token de acesso ao recurso compartilhado. O processo destinatário, configura seus atributos para obter acesso ao recurso, representado por um arquivo em uma pasta compartilhada entre os contêineres. O processo escreve no arquivo, imprime a saída escrita e repassa o token de acesso ao seu vizinho na rede.

## 5.3 Eleição de Líder: Algoritmo do Anel

O algoritmo do anel, implementa a classe Queue, que define uma fila para armazenar identificadores de nós. Um dos processos inicia a eleição de líder, utilizando novamente threads de cliente e servidor, para realizar comunicação sequencial e circular entre vizinhos na rede. O processo remetente elabora a mensagem de eleição, com alguns atributos do processo e a fila de eleição de líder. O processo destinatário recebe a mensagem e verifica se a fila de eleição está cheia. Se sim,

ele extrai o maior identificador da fila e inicia o compartilhamento do coordenador com os outros processos. Se não, ele adiciona seu identificador a fila e a repassa aos outros nós. A saída apresenta o estado da fila de eleição a cada envio de mensagem, o identificador do coordenador eleito e uma mensagem de sucesso no compartilhamento do líder com os outros processos.

Mais detalhes acerca da implementação e da execução dos algoritmos, podem ser vistos no repositório git em `https://github.com/dansc00/distributed-systems-project-unicamp`.