Computação Distribuída
Trabalho Prático: Exclusão Mútua

Bernard P. Ferreira[1]

Fernando Wagner Gomes Salim[1]

[1] Instituto de Ciências Exatas e Informática - Pontifícia Universidade Católica de Minas Gerais (PUC-MG) - Belo Horizonte - MG - Brasil

Introdução

A coordenação entre processos concorrentes em sistemas distribuídos é um dos desafios centrais da Computação Distribuída. Um dos aspectos fundamentais desse desafio é a exclusão mútua, que garante que múltiplos processos não acessem simultaneamente um recurso compartilhado. Diferentemente de sistemas centralizados, onde um monitor ou servidor pode arbitrar o acesso, em ambientes distribuídos não se pode pressupor a existência de um nó central confiável. Assim, é necessário que os próprios processos coordenem entre si a entrada na região crítica.

Dentre os algoritmos clássicos para exclusão mútua distribuída, o proposto por Ricart e Agrawala (1981) se destaca por sua simplicidade e eficiência. Baseado em troca de mensagens ponto a ponto, o algoritmo garante a exclusão mútua, o ordenamento das requisições e a ausência de deadlocks, utilizando apenas timestamps e identificadores únicos de processos.

Este trabalho tem como objetivo implementar uma simulação prática do algoritmo de Ricart e Agrawala em um sistema com múltiplos nós, demonstrando seu funcionamento por meio de logs e acessos coordenados a um recurso compartilhado simulado: um servidor de impressão.

Metodologia

A implementação foi desenvolvida em Python 3.12, escolhida por sua clareza sintática e suporte a bibliotecas de rede e concorrência. O sistema é composto por três módulos principais:

- cliente.py: módulo responsável por executar a lógica do algoritmo Ricart e Agrawala em cada nó. Cada cliente gerencia seu estado (RELEASED, WANTED ou HELD), envia mensagens de requisição e resposta, e realiza o controle do acesso à seção crítica.
- servidor.py: módulo que escuta conexões TCP recebidas por um nó. Ele interpreta mensagens recebidas de outros clientes e repassa ao cliente local as requisições ou confirmações.
- servidor_impressao.py: simula o recurso compartilhado, imprimindo na tela os valores

enviados por um processo autorizado a acessar a seção crítica.

A comunicação entre os processos foi realizada via sockets TCP/IP, e o controle da seção crítica é completamente descentralizado: nenhum componente do sistema decide quem imprime, apenas os próprios clientes coordenam entre si por meio de trocas de mensagens.

Ambiente de Execução

A execução foi realizada em um único computador, simulando múltiplos nós locais com endereços distintos via porta TCP. As configurações do ambiente são:

- Sistema Operacional: Microsoft Windows 11 Home
- Processador: Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz
- Memória RAM: 16GB
- Linguagem: Python 3.12
- Execução: 2 a 6 nós simulados simultaneamente, cada um em uma thread separada

Funcionamento do Código e Execução

O sistema desenvolvido simula a exclusão mútua em um ambiente distribuído utilizando o algoritmo proposto por Ricart e Agrawala. O código é dividido em quatro arquivos principais: cliente.py, servidor.py, servidor_impressao.py e main.py.

O arquivo main.py é o ponto de entrada do sistema. Ao executá-lo, o usuário informa o número de nós desejados. Em seguida, o sistema cria uma configuração com endereços locais (localhost) e portas distintas para cada nó. O servidor de impressão é iniciado em uma thread separada escutando na porta 6000, e cada nó inicia dois módulos: um Cliente e um Servidor.

O Cliente implementa o algoritmo de Ricart e Agrawala, sendo responsável por solicitar acesso à seção crítica, aguardar mensagens de confirmação dos demais nós e controlar o acesso ao recurso compartilhado. Quando entra na seção crítica, o cliente envia uma sequência numérica para o servidor de impressão, respeitando o intervalo de 0.5 segundos entre cada envio. Esse envio é feito número a número, garantindo que apenas um processo acesse o recurso por vez.

O Servidor associado a cada nó escuta mensagens de outros clientes. Quando recebe uma requisição, ele decide se responde imediatamente com 'ok' ou se posterga a resposta com base na prioridade do timestamp e do ID do processo.

O Servidor de Impressão apenas imprime os números recebidos. Ele não participa do controle de acesso, sendo um recurso passivo. Isso garante que o controle da exclusão mútua permaneça totalmente distribuído.

Para executar o sistema, basta executar o arquivo main.py, inserir o número de nós desejado, e

observar os logs no console. A exclusão mútua é mantida, e o comportamento do algoritmo pode ser verificado pela sequência ordenada e não intercalada das impressões no servidor central.

Resultados e Observações

A execução mostrou que o algoritmo de Ricart e Agrawala foi implementado corretamente. Os logs gerados durante a simulação revelam que:

- Cada cliente solicita acesso à seção crítica de forma assíncrona
- O processo só entra na seção crítica após receber mensagens ok de todos os outros nós
- Durante o período em que um cliente está na seção crítica, nenhum outro cliente imprime

A ausência de intercalamento entre impressões de processos distintos demonstra que a exclusão mútua está sendo respeitada. Como cada cliente realiza a impressão passo a passo dentro da seção crítica (com sleep(0.5) no cliente, não no servidor), o controle do acesso ao recurso é efetivo e distribuído.

Conclusão

A implementação proposta atingiu com êxito os objetivos definidos. O algoritmo de Ricart e Agrawala foi corretamente reproduzido em ambiente simulado, sem dependência de controle central. O uso de sockets TCP e controle de acesso assíncrono mostrou-se suficiente para coordenar múltiplos processos, cada um decidindo de forma autônoma quando requisitar e liberar o acesso ao recurso compartilhado.

Como melhoria futura, seria possível implementar um controle mais sofisticado do tempo de resposta com relógios vetoriais, bem como introduzir tolerância a falhas (por exemplo, detecção de queda de nós).

Referências

1. Ricart, G., & Agrawala, A. K. (1981). An optimal algorithm for mutual exclusion in computer networks. Communications of the ACM, 24(1), 9–17.
2. Tanenbaum, A. S., & Van Steen, M. (2007). Distributed Systems: Principles and Paradigms. 2ª ed. Pearson.
3. Python Socket Programming — https://docs.python.org/3/library/socket.html