Trabalho 2 - MC714 - 1º semestre/2024

Larissa Souto Zagati - RA 178060

Introdução

Este trabalho se concentra na implementação de algoritmos para resolução de problemas em sistemas distribuídos, utilizando a comunicação via troca de mensagens com MPI (Message Passing Interface). A escolha desses algoritmos visa explorar conceitos fundamentais em sistemas distribuídos, tais como sincronização de tempo, exclusão mútua e eleição de líder, proporcionando um entendimento prático e aprofundado desses tópicos.
Os algoritmos implementados foram: **Relógio Lógico de Lamport, Algoritmo de Exclusão Mútua e Algoritmo de Eleição de Líder.** A implementação foi realizada utilizando a biblioteca MPI para Python, conhecida como mpi4py, que permite a comunicação eficiente entre processos distribuídos.

Execução

Para execução dos algoritmos, basta rodar o comando:
docker compose up –detach –build

Relógio Lógico de Lamport

O problema do relógio lógico de Lamport aparece em sistemas distribuídos, onde cada processo tem seu próprio relógio e não há um relógio central para todos. Isso pode causar confusão na ordem dos eventos. O relógio lógico de Lamport ajuda a ordenar os eventos de forma consistente, mesmo sem um relógio global. A ideia é que cada processo tenha um contador inteiro (relógio lógico) que é incrementado em certos eventos. As regras principais são:
Incremento Local: O relógio lógico é incrementado em 1 antes de um evento local (como enviar uma mensagem).
Envio de Mensagens: Quando um processo envia uma mensagem, ele inclui seu relógio lógico na mensagem.
Recepção de Mensagens: Quando um processo recebe uma mensagem, ele ajusta seu relógio lógico para ser o maior entre o valor atual do relógio lógico e o valor do relógio lógico

recebido na mensagem, incrementado em 1. Isso garante que, se um evento A acontece antes de um evento B em um processo, o valor do relógio lógico de A será menor que o valor do relógio lógico de B.

Detalhes da implementação

- Inicialização de variáveis:
  comm: objeto de comunicação MPI.
  rank: identificador do processo.
  size: número total de processos.
  logical_clock: relógio lógico inicializado para cada processo.

- Função de envio de mensagem (send_message): incrementa o relógio lógico e envia a mensagem contendo o relógio lógico e o rank do processo.

- Função de recebimento de mensagem (receive_message): recebe a mensagem e atualiza o relógio lógico usando a regra do máximo entre o relógio atual e o recebido, incrementado em 1.

- Na main: o processo com rank 0 envia mensagens para todos os outros processos, com um intervalo aleatório para simular assincronismo, os outros processos recebem a mensagem do processo 0. A chamada de comm.Barrier() serve para sincronizar todos os processos antes de finalizar, garantindo que todos os processos tenham recebido e processado as mensagens.

A implementação do relógio lógico de Lamport com MPI em Python foi escolhida por diversas razões que destacam sua simplicidade, eficiência e clareza no contexto de sistemas distribuídos. Esta implementação mostra como usar relógios lógicos de Lamport para manter uma ordenação consistente de eventos em um sistema distribuído.

Testes realizados

Foram feitos testes com dois, três e quatro processos simulando envio e recepção de mensagens e verificado se os relógios lógicos são atualizados corretamente conforme as mensagens são enviadas e recebidas.
Exemplo de resultado com quatro processos:

```
mc714-trabalho2-178060-lamport_clock-2      | Processo 0 enviou mensagem para o processo 1. Clock: 1
mc714-trabalho2-178060-lamport_clock-2      | Processo 0 enviou mensagem para o processo 2. Clock: 2
mc714-trabalho2-178060-lamport_clock-2      | Processo 0 enviou mensagem para o processo 3. Clock: 3
mc714-trabalho2-178060-lamport_clock-2      | Processo 0 - Clock final: 3
mc714-trabalho2-178060-lamport_clock-2      | Processo 1 recebeu mensagem do processo 0. Clock: 2
mc714-trabalho2-178060-lamport_clock-2      | Processo 1 - Clock final: 2
mc714-trabalho2-178060-lamport_clock-2      | Processo 2 recebeu mensagem do processo 0. Clock: 3
mc714-trabalho2-178060-lamport_clock-2      | Processo 2 - Clock final: 3
mc714-trabalho2-178060-lamport_clock-2      | Processo 3 recebeu mensagem do processo 0. Clock: 4
mc714-trabalho2-178060-lamport_clock-2      | Processo 3 - Clock final: 4
mc714-trabalho2-178060-lamport_clock-2 exited with code 0
mc714-trabalho2-178060-lamport_clock-4      | Processo 0 enviou mensagem para o processo 1. Clock: 1
mc714-trabalho2-178060-lamport_clock-4      | Processo 0 enviou mensagem para o processo 2. Clock: 2
mc714-trabalho2-178060-lamport_clock-4      | Processo 0 enviou mensagem para o processo 3. Clock: 3
mc714-trabalho2-178060-lamport_clock-4      | Processo 0 - Clock final: 3
mc714-trabalho2-178060-lamport_clock-4      | Processo 1 recebeu mensagem do processo 0. Clock: 2
mc714-trabalho2-178060-lamport_clock-4      | Processo 1 - Clock final: 2
mc714-trabalho2-178060-lamport_clock-4      | Processo 2 recebeu mensagem do processo 0. Clock: 3
mc714-trabalho2-178060-lamport_clock-4      | Processo 2 - Clock final: 3
mc714-trabalho2-178060-lamport_clock-4      | Processo 3 recebeu mensagem do processo 0. Clock: 4
mc714-trabalho2-178060-lamport_clock-4      | Processo 3 - Clock final: 4
mc714-trabalho2-178060-lamport_clock-4 exited with code 0
mc714-trabalho2-178060-lamport_clock-1      | Processo 0 enviou mensagem para o processo 1. Clock: 1
mc714-trabalho2-178060-lamport_clock-1      | Processo 0 enviou mensagem para o processo 2. Clock: 2
mc714-trabalho2-178060-lamport_clock-1      | Processo 0 enviou mensagem para o processo 3. Clock: 3
mc714-trabalho2-178060-lamport_clock-1      | Processo 0 - Clock final: 3
mc714-trabalho2-178060-lamport_clock-1      | Processo 1 recebeu mensagem do processo 0. Clock: 2
mc714-trabalho2-178060-lamport_clock-1      | Processo 1 - Clock final: 2
mc714-trabalho2-178060-lamport_clock-1      | Processo 2 recebeu mensagem do processo 0. Clock: 3
mc714-trabalho2-178060-lamport_clock-1      | Processo 2 - Clock final: 3
mc714-trabalho2-178060-lamport_clock-1      | Processo 3 recebeu mensagem do processo 0. Clock: 4
mc714-trabalho2-178060-lamport_clock-1      | Processo 3 - Clock final: 4
mc714-trabalho2-178060-lamport_clock-1 exited with code 0
```

Exclusão Mútua

Exclusão mútua é um problema em sistemas distribuídos onde vários processos precisam acessar um recurso compartilhado de forma coordenada para evitar conflitos ou inconsistências. Basicamente, a ideia é garantir que apenas um processo por vez possa acessar a seção crítica (a parte do código que manipula o recurso compartilhado).

Detalhes da implementação utilizando token

O código utiliza MPI para coordenar a exclusão mútua entre processos distribuídos.

- Inicialização de variáveis:
  token: indica se o processo possui o token de exclusão mútua.
  in_cs: indica se o processo está na seção crítica.
  lamport_clock: relógio de Lamport usado para sincronização de eventos.
  request_queue: fila de solicitações para entrar na seção crítica, armazenando o número do processo e o valor do relógio de Lamport quando a solicitação foi feita.

- Função que envia mensagens (send_message): envia uma mensagem para o destino especificado com uma determinada tag e incrementa o relógio antes de enviar.

- Função de recebimento de mensagem (receive_message): recebe uma mensagem de qualquer fonte e atualiza o relógio com o timestamp recebido.

- Função request_cs: o processo atual adiciona sua própria solicitação à request_queue. Enviamos mensagens de solicitação para todos os outros processos. O processo aguarda até receber o token. Quando o token é recebido, verificamos se o processo ainda está na fila (request_queue) e removemos sua própria solicitação. O processo entra na seção crítica (in_cs é definido como True).

- Função release_cs: o processo atual imprime uma mensagem indicando que está saindo da seção crítica e mostra o valor atual do relógio. O processo atual libera a seção crítica (in_cs é definido como False). O token é passado para o próximo processo na fila (next_process).

- Na main: executa um loop onde cada processo realiza trabalho na seção não crítica, faz requisições para entrar na seção crítica se necessário, e espera pelo token para entrar na seção crítica quando disponível.

Essa implementação foi escolhida por ser simples e clara, o token é um jeito eficiente de garantir que só um processo por vez acesse o recurso compartilhado, evitando conflitos e problemas em sistemas distribuídos.

Testes realizados

A implementação foi executada em um ambiente MPI com múltiplos processos para verificar se todos os processos conseguem enviar e receber mensagens corretamente. Também foi verificado se cada processo entra e sai da seção crítica de forma ordenada, sem que dois processos entrem na seção crítica ao mesmo tempo. Além disso, também foi checado se os processos respondem corretamente às requisições de entrada na seção crítica, adicionando a requisição à fila quando necessário e enviando respostas apropriadas. Por fim, a sincronização do relógio lógico foi monitorada para garantir que os timestamps são atualizados corretamente com base nas mensagens enviadas e recebidas.
Exemplo de resultado com três processos:

```
mc714-trabalho2-178060-mutual_exclusion-1  | Processo 0 entrou na seção crítica. Clock: 3
mc714-trabalho2-178060-mutual_exclusion-1  | Processo 0 saiu da seção crítica. Clock: 3
mc714-trabalho2-178060-mutual_exclusion-1  | Processo 0 entrou na seção crítica. Clock: 15
mc714-trabalho2-178060-mutual_exclusion-1  | Processo 0 saiu da seção crítica. Clock: 15
mc714-trabalho2-178060-mutual_exclusion-1  | Processo 0 entrou na seção crítica. Clock: 26
mc714-trabalho2-178060-mutual_exclusion-1  | Processo 0 saiu da seção crítica. Clock: 26
mc714-trabalho2-178060-mutual_exclusion-1  | Processo 0 entrou na seção crítica. Clock: 37
mc714-trabalho2-178060-mutual_exclusion-1  | Processo 0 saiu da seção crítica. Clock: 37
mc714-trabalho2-178060-mutual_exclusion-1  | Processo 0 entrou na seção crítica. Clock: 48
mc714-trabalho2-178060-mutual_exclusion-1  | Processo 0 saiu da seção crítica. Clock: 48
mc714-trabalho2-178060-mutual_exclusion-1  | Processo 1 entrou na seção crítica. Clock: 7
mc714-trabalho2-178060-mutual_exclusion-1  | Processo 1 saiu da seção crítica. Clock: 7
mc714-trabalho2-178060-mutual_exclusion-1  | Processo 1 entrou na seção crítica. Clock: 18
mc714-trabalho2-178060-mutual_exclusion-1  | Processo 1 saiu da seção crítica. Clock: 18
mc714-trabalho2-178060-mutual_exclusion-1  | Processo 1 entrou na seção crítica. Clock: 29
mc714-trabalho2-178060-mutual_exclusion-1  | Processo 1 saiu da seção crítica. Clock: 29
mc714-trabalho2-178060-mutual_exclusion-1  | Processo 1 entrou na seção crítica. Clock: 40
mc714-trabalho2-178060-mutual_exclusion-1  | Processo 1 saiu da seção crítica. Clock: 40
mc714-trabalho2-178060-mutual_exclusion-1  | Processo 1 entrou na seção crítica. Clock: 51
mc714-trabalho2-178060-mutual_exclusion-1  | Processo 1 saiu da seção crítica. Clock: 51
mc714-trabalho2-178060-mutual_exclusion-1  | Processo 2 entrou na seção crítica. Clock: 9
mc714-trabalho2-178060-mutual_exclusion-1  | Processo 2 saiu da seção crítica. Clock: 9
mc714-trabalho2-178060-mutual_exclusion-1  | Processo 2 entrou na seção crítica. Clock: 20
```

Eleição de Líder

A eleição de líder é um problema em sistemas distribuídos para garantir que apenas um processo seja responsável por coordenar atividades críticas, como evitar conflitos de escrita em bases de dados distribuídas ou coordenar ações entre múltiplos nós. O problema geralmente envolve garantir que todos os processos em um sistema distribuído cheguem a um consenso sobre quem será o líder.

Detalhes da implementação utilizando algoritmo em anel
Esta implementação foi retirada do algoritmo de eleição em anel do site geeksforgeeks (link nas referências) e modificado de forma a incluir a comunicação MPI.

- Inicialização (initialiseRing): aqui, a classe Ring_Election inicializa o número total de processos (TotalProcess) usando comm.Get_size() do MPI. Cada processo é representado pela classe Pro, que armazena um identificador (id) e um status de atividade (act).

- Função de eleição (Election): para simular a falha de um processo, um processo aleatório é marcado como inativo (act = False). Isso introduz a possibilidade de falhas no sistema distribuído - uma realidade comum em ambientes reais. Em seguida, um processo é escolhido aleatoriamente para iniciar a eleição (initializedProcess). Em um ambiente real, este passo poderia ser determinado por algum critério pré-determinado, como um processo que percebe a falha do líder atual. O processo que inicia a eleição passa um token para o próximo processo na ordem do anel. Esse processo continua até que o token retorne ao processo inicial, indicando que todos os processos ativos foram consultados. Após o token circular, o processo ativo com o maior identificador (id) é escolhido como coordenador. Isso garante que o coordenador tenha a maior prioridade

entre os processos ativos. Após a eleição do coordenador, ele passa uma mensagem para todos os processos ativos no anel. Isso informa a todos os processos quem é o novo líder, facilitando a coordenação futura.

- Máximo Identificador Ativo (FetchMaximum): esta função auxilia a encontra o processo ativo com o maior identificador, que é utilizado na fase de escolha do coordenador.

- Na main: aqui, o programa MPI é iniciado, cada processo MPI é inicializado como um objeto Ring_Election, e a eleição é iniciada.

Essa implementação foi escolhida porque o algoritmo de eleição em anel é simples de entender e implementar em sistemas distribuídos usando MPI. Ele é eficaz para sistemas onde cada processo pode se comunicar diretamente apenas com seus vizinhos imediatos no anel.

Testes realizados

Foi executado o teste de eleição, o teste de concorrência para simular várias eleições ocorrendo ao mesmo tempo para ver se o sistema resolve os conflitos de eleição corretamente e um teste de recuperação de falha que simula a falha do líder para verificar se o sistema consegue se recuperar e manter a operação correta.
Exemplo de resultado da eleição:

```
mc714-trabalho2-178060-leader_election-1   | Processo 1 falha
mc714-trabalho2-178060-leader_election-1   | Eleicao iniciada por 0
mc714-trabalho2-178060-leader_election-1   | Processo 0 passa Eleição(0) para 2
mc714-trabalho2-178060-leader_election-1   | Processo 2 passa Eleição(2) para 3
mc714-trabalho2-178060-leader_election-1   | Processo 3 torna-se coordenador
mc714-trabalho2-178060-leader_election-1   | Processo 3 passa mensagem Coordenador(3) para processo 0
mc714-trabalho2-178060-leader_election-1   | Processo 0 passa mensagem Coordenador(3) para processo 2
mc714-trabalho2-178060-leader_election-1   | Fim da Eleicao
mc714-trabalho2-178060-leader_election-1   | Processo 2 falha
mc714-trabalho2-178060-leader_election-1   | Eleicao iniciada por 2
mc714-trabalho2-178060-leader_election-1   | Processo 2 passa Eleição(2) para 3
mc714-trabalho2-178060-leader_election-1   | Processo 3 passa Eleição(3) para 0
mc714-trabalho2-178060-leader_election-1   | Processo 0 passa Eleição(0) para 1
mc714-trabalho2-178060-leader_election-1   | Processo 3 torna-se coordenador
mc714-trabalho2-178060-leader_election-1   | Processo 3 passa mensagem Coordenador(3) para processo 0
mc714-trabalho2-178060-leader_election-1   | Processo 0 passa mensagem Coordenador(3) para processo 1
mc714-trabalho2-178060-leader_election-1   | Fim da Eleicao
mc714-trabalho2-178060-leader_election-1   | Processo 2 falha
mc714-trabalho2-178060-leader_election-1   | Eleicao iniciada por 1
mc714-trabalho2-178060-leader_election-1   | Processo 1 passa Eleição(1) para 3
mc714-trabalho2-178060-leader_election-1   | Processo 3 passa Eleição(3) para 0
mc714-trabalho2-178060-leader_election-1   | Processo 3 torna-se coordenador
mc714-trabalho2-178060-leader_election-1   | Processo 3 passa mensagem Coordenador(3) para processo 0
mc714-trabalho2-178060-leader_election-1   | Processo 0 passa mensagem Coordenador(3) para processo 1
mc714-trabalho2-178060-leader_election-1   | Fim da Eleicao
mc714-trabalho2-178060-leader_election-1   | Processo 1 falha
mc714-trabalho2-178060-leader_election-1   | Eleicao iniciada por 1
mc714-trabalho2-178060-leader_election-1   | Processo 1 passa Eleição(1) para 2
mc714-trabalho2-178060-leader_election-1   | Processo 2 passa Eleição(2) para 3
mc714-trabalho2-178060-leader_election-1   | Processo 3 passa Eleição(3) para 0
mc714-trabalho2-178060-leader_election-1   | Processo 3 torna-se coordenador
mc714-trabalho2-178060-leader_election-1   | Processo 3 passa mensagem Coordenador(3) para processo 0
mc714-trabalho2-178060-leader_election-1   | Processo 0 passa mensagem Coordenador(3) para processo 2
mc714-trabalho2-178060-leader_election-1   | Fim da Eleicao
```

Referências

https://mpi4py.readthedocs.io/en/stable/overview.html
https://pt.wikipedia.org/wiki/Rel%C3%B3gios_de_Lamport
https://www.geeksforgeeks.org/mutual-exclusion-in-distributed-system/
https://pt.wikipedia.org/wiki/Exclus%C3%A3o_m%C3%BAtua
https://www.geeksforgeeks.org/election-algorithm-and-distributed-processing/
GPT-4