Exclusão Mútua e Eleição de Líder

Raquel Mini - raquelmini@pucminas.br

Exclusão Mútua Distribuída

Os processos distribuídos frequentemente precisam coordenar suas atividades

Se um conjunto de processos compartilham um recurso então frequentemente a exclusão mútua é exigida

Precisamos de uma solução para seção crítica que seja baseada unicamente na passagem de mensagens

Exclusão Mútua Distribuída

Consideremos um sistema de N processos p_i, $i = 1, 2, ..., N$ que não compartilham variáveis

Os processos acessam recursos comuns e fazem isso em uma seção crítica

Suposições:

- Sistema é assíncrono
- Processos não falham
- Envio de mensagens é confiável

Exclusão Mútua Distribuída

O protocolo em nível de aplicação para executar a seção crítica é o seguinte:

enter()	// entra na seção crítica – bloqueia, se necessário
resourceAccesses()	// acessa recursos compartilhados na seção crítica
exit()	// sai da seção crítica – outros processos podem entrar agora

Requisitos básicos são os seguintes:

EM1: (segurança)	No máximo um processo por vez pode ser executado na seção crítica (SC).
EM2: (subsistência)	As requisições para entrar e sair da seção crítica têm sucesso.

Exclusão Mútua Distribuída

A condição EM2 implica em independência de impasse (*deadlock*) e inanição (*starvation*)

- Impasse: quando dois ou mais processos estão travados indefinidamente enquanto tentam entrar ou sair da seção crítica, devido à sua interdependência mútua
- Inanição: o adiamento indefinido da entrada de um processo que a solicitou

Exclusão Mútua Distribuída

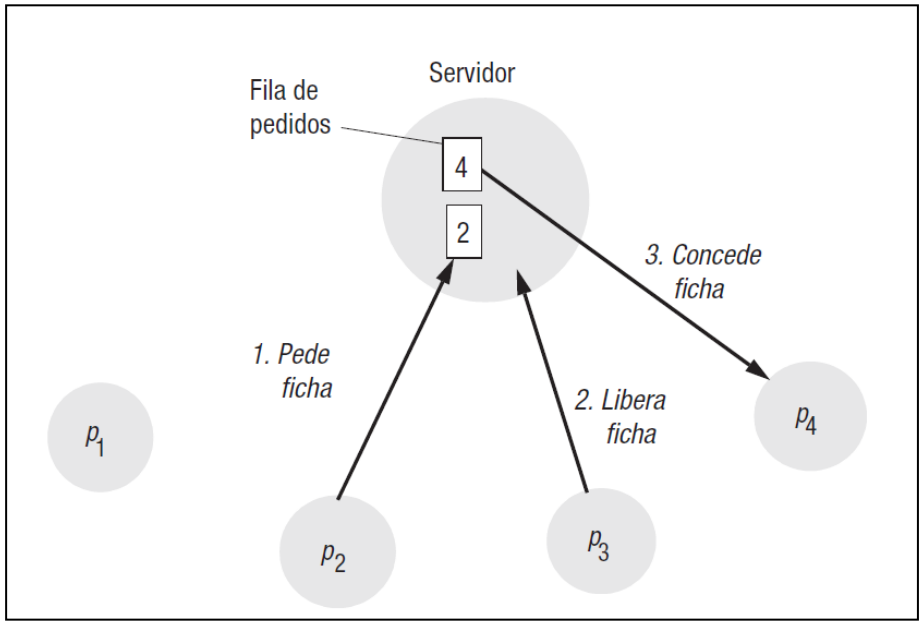
Como garantir a imparcialidade no acesso à seção crítica?

- Não é possível ordenar a entrada na seção crítica pelos tempos que os processos a solicitaram, devido à ausência de relógios globais
- Podemos usar a relação ordem acontece antes entre as mensagens que solicitam a entrada na seção crítica

EM3: (ordenação $\rightarrow$) Se uma requisição para entrar na SC aconteceu antes de outra, então a entrada na SC é garantida nessa ordem.

Algoritmo do Servidor Central

Um servidor concede permissão para entrar na seção crítica



Algoritmo do Servidor Central

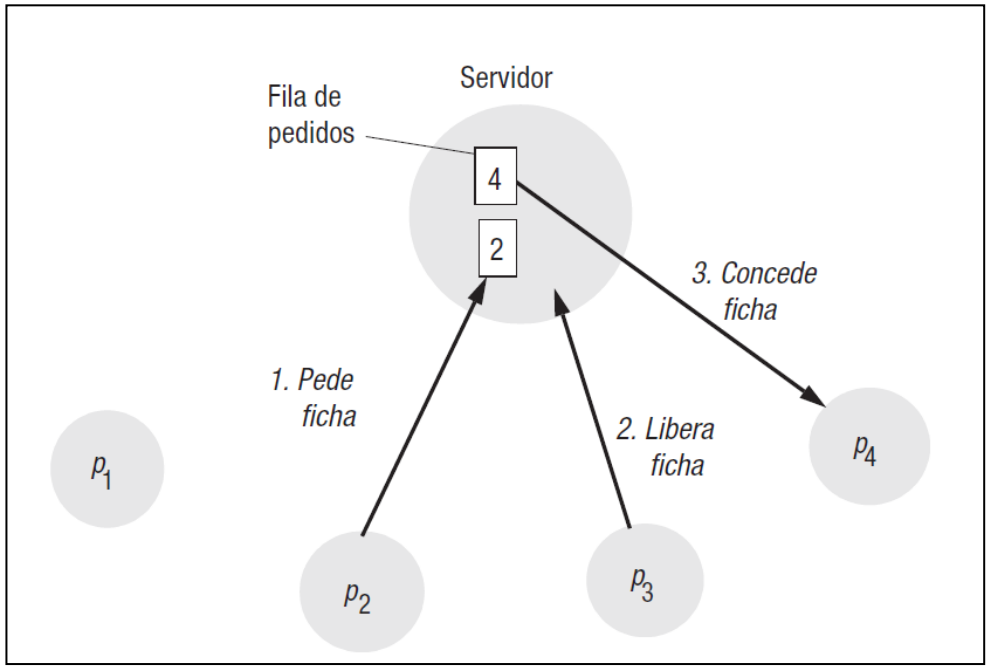
Para entrar na seção crítica, um processo envia uma mensagem de requisição para o servidor e espera uma resposta

- A resposta constitui uma ficha (*token*) significando permissão para entrar na seção crítica
- Se nenhum outro processo tiver a ficha no momento da requisição, então o servidor responderá imediatamente, concedendo a ficha

Algoritmo do Servidor Central

- Se a ficha estiver de posse de outro processo, então o servidor não responderá, mas enfileirará a requisição
- Na saída da seção crítica, uma mensagem é enviada para o servidor devolvendo a ficha a ele
- Se a fila de processos em espera não estiver vazia, o servidor escolherá a entrada mais antiga, a removerá e responderá para o processo correspondente
- O processo escolhido terá a posse da ficha

Algoritmo do Servidor Central



O servidor pode se tornar um gargalo de desempenho para o sistema como um todo

Algoritmo Baseado em Anel

Não exige um processo adicional

Supõe que os processos são organizados como um anel lógico (*token ring*), por onde circula um *token*

- Não exige que a topologia da rede seja em anel (anel lógico e não físico)
- Cada processo p_i tem um canal de comunicação com o processo seguinte no anel

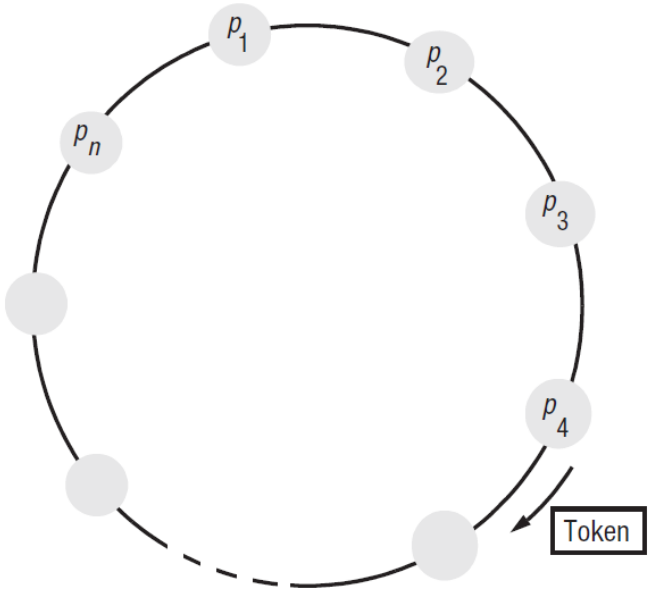
Algoritmo Baseado em Anel

A exclusão é concedida pela obtenção do *token* na forma de uma mensagem passada de processo para processo, em uma única direção em torno do anel

- Se um processo não deseja entrar na seção crítica:
  - Ao receber o *token*, repassa a mesma para seu vizinho
- Se um processo deseja entrar na seção crítica:
  - Aguarda a passagem da *token* e a retém
- Para sair da seção crítica, o processo envia o *token* para seu vizinho

Algoritmo Baseado em Anel

As condições EM1 e EM2 são satisfeitas por esse algoritmo, mas o *token* não é necessariamente obtido na ordem acontece antes



Algoritmo Baseado em Anel

Vantagens: simplicidade

Desvantagem:

- Os processos enviam mensagens em torno do anel, mesmo quando nenhum processo solicita entrada na seção crítica

Algoritmo de Ricart e Agrawala

Algoritmo para implementar exclusão mútua entre N processos baseado em *multicast*

Os processos que solicitam a entrada em uma seção crítica difundem seletivamente (*multicast*) uma mensagem de requisição e só podem entrar nela quando todos os outros processos tiverem respondido a essa mensagem

É necessário garantir as condições EM1, EM2 e EM3

Algoritmo de Ricart e Agrawala

Os processos p_1, p_2,...p_N apresentam identificadores numéricos distintos

As mensagens que solicitam entrada são da forma $<T, p_i>$ onde T é o carimbo de tempo do remetente e p_i é o identificador do remetente

Cada processo registra seu estado em uma variável *state*

- RELEASED: fora da seção crítica
- WANTED: querendo entrar
- HELD: dentro da seção crítica

Algoritmo de Ricart e Agrawala

Na inicialização

state := RELEASED;

Para entrar na seção

state := WANTED;
Envia a requisição por *multicast* para todos os processos;
T := carimbo de tempo da requisição;

} *Processamento da requisição referido aqui*

Espera até (número de respostas recebidas = (N – 1));
state := HELD;

No recebimento de uma requisição $<T_i, p_i>$ *em* p_j $(i \neq j)$

if (*state* = HELD *or* (*state* = WANTED *and* $(T, p_j) < (T_i, p_i)$))
then
enfileira requisição de p_i sem responder;
else
responde imediatamente para p_i;
end if

Para sair da seção crítica

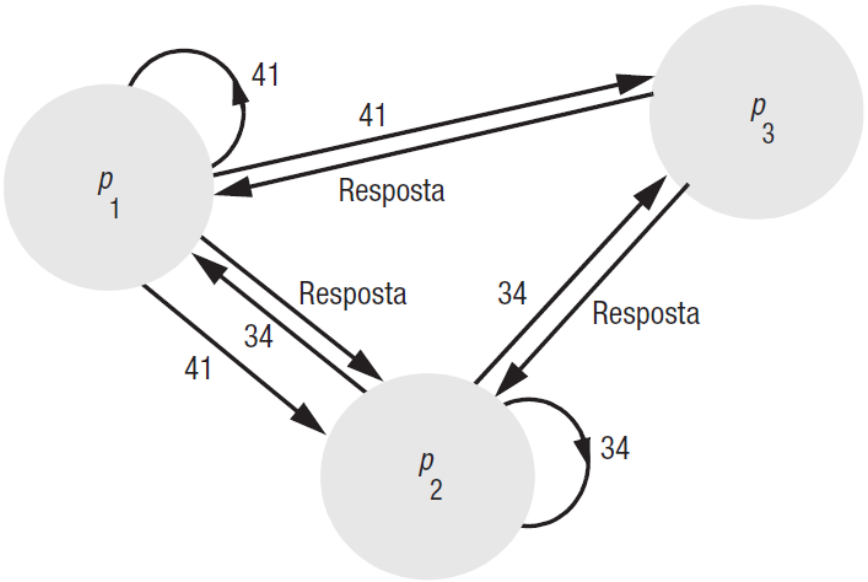
state := RELEASED;
responde a todas as requisições enfileiradas;

Algoritmo de Ricart e Agrawala

p_3 não está interessado em entrar na seção crítica

p_1 e p_2 solicitam entrada na seção crítica

- Carimbo de tempo de p_1 é 41
- Carimbo de tempo de p_2 é 34



Algoritmo de Ricart e Agrawala

Número de mensagens trocadas para entrar na seção crítica: $2\ (n-1)$

- $n-1$: *requests*
- $n-1$: *replies*

Vantagem:

- Distribuído (sem um ponto central de falha)

Algoritmo de Maekawa

Para um processo entrar em uma seção crítica, não é necessário que todos os seus pares concedam o acesso

Os processos só precisam obter permissão de subconjuntos de seus pares para entrar, desde que os subconjuntos usados por quaisquer dois processos se sobreponham

Podemos considerar os processos como votando um no outro para entrar na seção crítica

Um processo “candidato” deve reunir votos suficientes para entrar

Algoritmo de Maekawa

Os processos na interseção de dois conjuntos de votantes garantem a propriedade de segurança EM1, de que no máximo um processo pode entrar na seção crítica

Maekawa associou um conjunto de votação V_i a cada processo p_i ($i = 1, 2..., N$), onde $V_i \subseteq \{p_1, p_1..., p_N\}$. Os conjuntos V_i são escolhidos de modo que, para todo $i, j = 1, 2..., N$:

- $p_i \in V_i$;
- $V_i \cap V_j \neq \varnothing$: existe pelo menos um membro em comum de quaisquer dois conjuntos votantes;
- $|V_i| = K$: para ser imparcial, cada processo tem um conjunto votante de mesmo tamanho;
- Cada processo p_j está contido em M conjuntos votantes V_i.

Algoritmo de Maekawa

Cada processo está em tantos conjuntos de votação quantos são os elementos em cada um desses conjuntos

Para obter a entrada na seção crítica, um processo p_i envia mensagens de requisição para todos os K membros de V_i (incluindo ele mesmo)

p_i não pode entrar na seção crítica até que tenha recebido todas as K mensagens de resposta

Algoritmo de Maekawa

Quando um processo p_j em V_i recebe a mensagem de requisição de p_i

- Envia uma mensagem de resposta imediatamente a não ser que seu estado seja HELD ou que já tenha respondido ("votado") desde a última vez que recebeu uma mensagem de liberação
- Caso contrário, ele enfileira a mensagem de requisição, mas não responde ainda

Algoritmo de Maekawa

Quando um processo recebe uma mensagem de liberação, ele remove o nó cabeça de sua fila de requisições pendentes e envia uma mensagem de resposta (um "voto")

Para sair da seção crítica, p_i envia mensagens de liberação para todos os K membros de V_i (incluindo ele mesmo)

Algoritmo de Maekawa

Na inicialização

state := RELEASED;

voted := FALSE;

Para p_i *entrar na seção crítica*

state := WANTED;

Envia a *requisição* por *multicast* para todos os processos em V_i;

Espera até (número de respostas recebidas = K);

state := HELD;

No recebimento de uma requisição de p_i *em* p_j

if (*state* = HELD *ou voted* = TRUE)

then

enfileira a requisição de p_i sem responder;

else

envia resposta para p_i;

voted := TRUE;

end if

Algoritmo de Maekawa

Para p_i sair da seção crítica

state := RELEASED;

Envia a *liberação* via *multicast* para todos os processos em V_i;

No recebimento de uma liberação de p_i em p_j

if (fila de requisições não estiver vazia)

then

remove cabeça da fila – digamos, p_k;

envia *resposta* para p_k;

voted := TRUE;

else

voted := FALSE;

end if

Tarefa 6 (parte 1) – postar no Canvas até 28/03/2021

1. No algoritmo do servidor central para exclusão mútua, descreva uma situação na qual duas requisições não são processadas na ordem acontece antes
2. Dê um exemplo de execução do algoritmo baseado em anel para mostrar que os processos não têm necessariamente a entrada garantida na seção crítica na ordem acontece antes.

Tarefa 6 (parte 1) – postar no Canvas até 28/03/2021

3. Em determinado sistema, cada processo normalmente usa uma seção crítica muitas vezes, antes que outro processo a solicite. Explique por que o algoritmo de exclusão mútua de Ricart e Agrawala é ineficiente para esse caso e descreva como fazer para melhorar seu desempenho.

Eleição de Líder

Algoritmo para escolher um único processo para desempenhar uma função em particular é chamado de algoritmo de eleição de líder

Cada processo p_i ($i = 1, 2, ..., N$) tem uma variável $eleito_i$, que conterá o identificador do processo eleito

Quando o processo se torna participante de uma eleição pela primeira vez, ele configura essa variável com o valor especial $\perp$ para denotar que ela ainda não está definida

Eleição de Líder

Os requisitos são que, durante qualquer execução em particular do algoritmo:

E1: (segurança)	Um processo participante p_i tem $eleito_i = \perp$ ou $eleito_i = P$, onde P é escolhido como o processo não defeituoso com o maior identificador no final da execução.
E2: (subsistência)	Todos os processos p_i participam e configuram $eleito_i \neq \perp$ ou falham.

Podem existir processos p_i que ainda não sejam participantes, os quais registram em $eleito_i$ o identificador do processo eleito anterior

Algoritmo de Eleição baseado em Anel

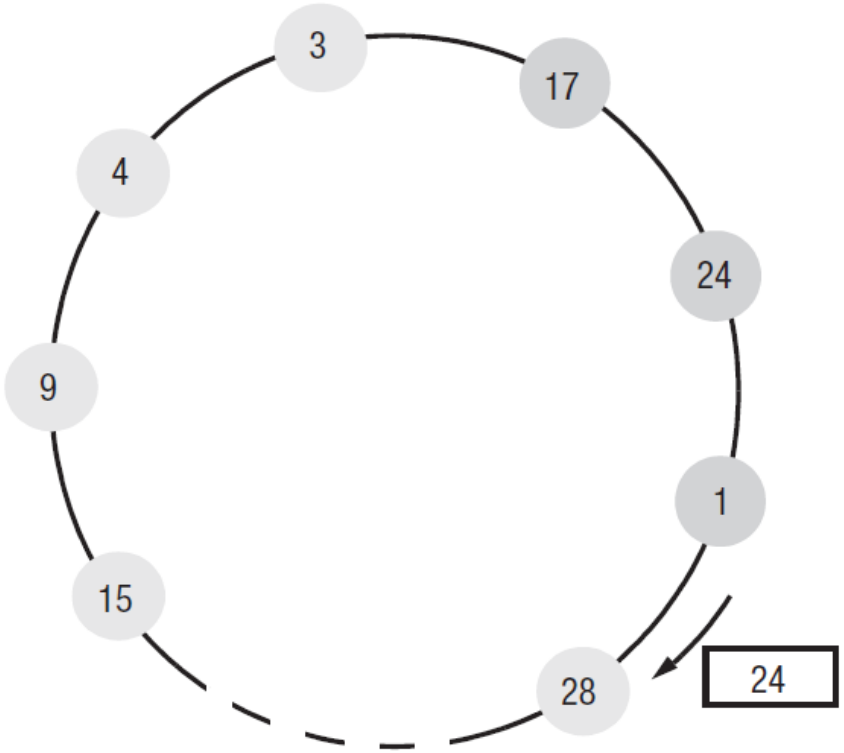
Cada processo p_i tem um canal de comunicação para o processo seguinte no anel $p_{(i+1) \bmod N}$

Todas as mensagens são enviadas no sentido horário em torno do anel

Supomos que não ocorrem falhas e que o sistema é assíncrono

O objetivo é eleger um único processo, chamado coordenador, que é aquele com o maior identificador

Algoritmo de Eleição baseado em Anel



Algoritmo de Eleição baseado em Anel

Inicialmente, cada processo é marcado como não participante de uma eleição

Qualquer processo pode iniciar uma eleição

- Neste caso, ele marca a si mesmo como participante, coloca seu identificador em uma mensagem de eleição e a envia para seu vizinho no sentido horário

Algoritmo de Eleição baseado em Anel

Quando um processo recebe uma mensagem de eleição, ele compara o identificador presente na mensagem com o seu próprio

- Se o identificador que chegou é maior, ele encaminha a mensagem para seu vizinho
- Se o identificador que chegou é menor e o receptor não é participante, ele substitui por seu próprio identificador na mensagem e a encaminha (ele não encaminha a mensagem se já for participante)
- No encaminhamento de uma mensagem de eleição, o processo marca a si mesmo como participante

Algoritmo de Eleição baseado em Anel

- Se o identificador recebido for o do próprio receptor, então ele se tornará o coordenador
- O coordenador marca a si mesmo como não participante e envia uma mensagem eleito para seu vizinho, anunciando seu eleição e incluindo sua identidade

Quando um processo p_i recebe uma mensagem eleito, ele

- marca a si mesmo como não participante
- configura sua variável $eleito_i$ com o identificador presente na mensagem
- se não for o novo coordenador, encaminha a mensagem para seu vizinho

Algoritmo de Eleição baseado em Anel

As condições E1 e E2 são satisfeitas

Se apenas um processo inicia uma eleição, então o pior caso se dá quando seu vizinho no sentido anti-horário tem o identificador mais alto

- Um total de N-1 mensagens é exigido para chegar a esse vizinho
- Esse vizinho não anunciará sua eleição até que seu identificador tenha completado outra volta (mais N mensagens)
- A mensagem eleito é então enviada N vezes
- Totalizando 3N – 1 mensagens

Algoritmo Valentão

Permite que os processos falhem durante uma eleição

Considera que a distribuição de mensagens entre os processos seja confiável

Presume que o sistema seja síncrono

- Usa tempos limites para detectar uma falha de processo

Presume que cada processo sabe quais processos têm identificadores mais altos e que pode se comunicar com todos esses processos

Algoritmo Valentão

Existem três tipos de mensagens

- Eleição: enviada para convocar uma eleição
- Resposta: resposta a uma mensagem de eleição
- Coordenador: enviada para anunciar a identidade do do processo eleito (novo coordenador)

Um processo inicia uma eleição quando observa, por meio de seus tempos limites, que o coordenador falhou (vários processos podem descobrir isso simultaneamente)

Algoritmo Valentão

O sistema é síncrono

- T_t: atraso máximo de transmissão de mensagem
- T_p: atraso máximo para processar uma mensagem
- $T = 2T_t + T_p$ é um limite superior para o tempo total decorrido desde o envio de uma mensagem até o outro processo receber uma resposta
- Se nenhuma mensagem chegar dentro do tempo T, o detector de falha local poderá relatar que o destinatário da requisição falhou

Algoritmo Valentão

O processo que sabe que possui o identificador mais alto pode eleger a si mesmo como coordenador

- Envia uma mensagem de coordenador para todos os processos com identificadores mais baixos

Algoritmo Valentão

Um processo com identificador mais baixo inicia uma eleição

- Envia uma mensagem de eleição para os processos que têm identificador mais alto, esperando receber uma mensagem de resposta em retorno
- Se nenhuma resposta chegar dentro do tempo T, o processo se considerará o coordenador
  - Enviará uma mensagem de coordenador para todos os processos com identificadores mais baixos, anunciando isso
- Se uma resposta chegar, o processo esperará por mais um período T' que uma mensagem de coordenador chegue do novo coordenador
- Se nenhuma mensagem chegar, ele iniciará outra eleição

Algoritmo Valentão

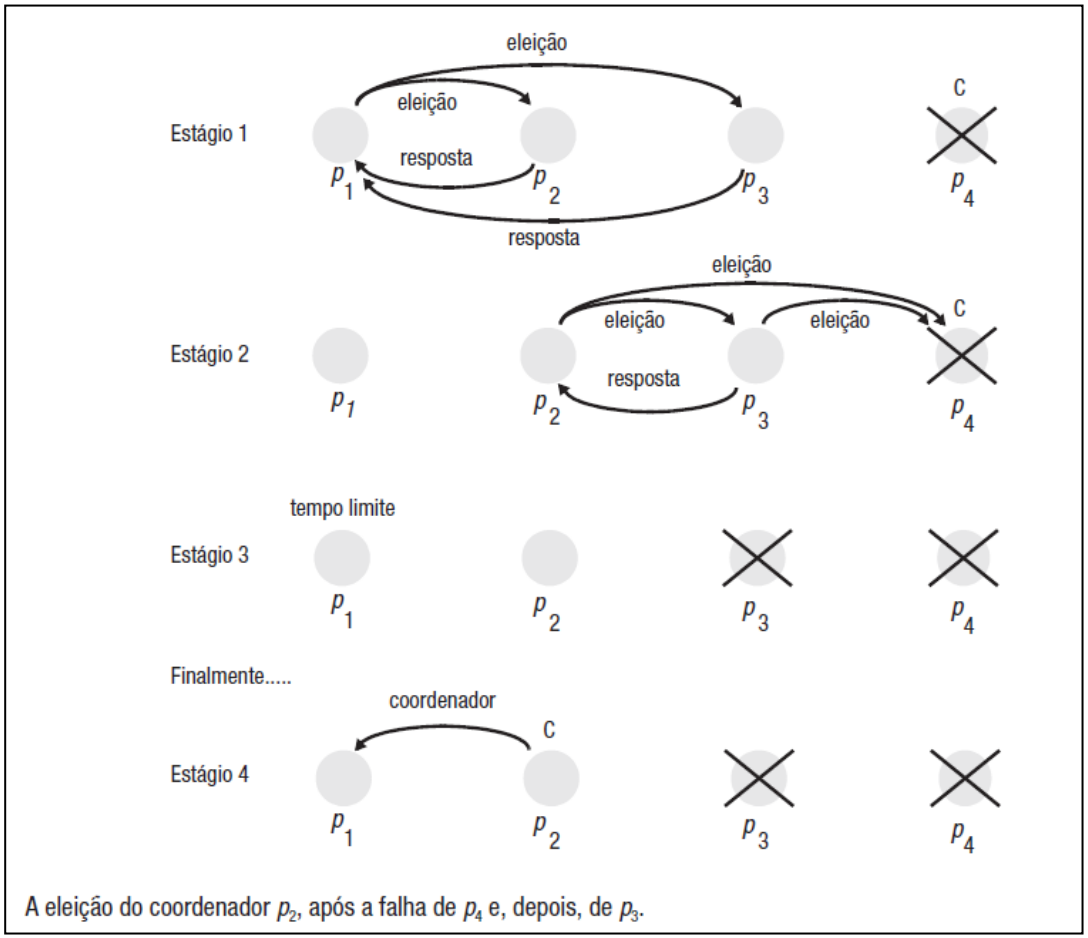
Se um processo p_i recebe uma mensagem de coordenador, ele configura sua variável $eleito_i$ com o identificador do coordenador contido dentro dela

Se um processo recebe uma mensagem de eleição, ele envia de volta uma mensagem de resposta e inicia outra eleição (a não ser que já tenha iniciado uma)

Quando um processo para substituir um processo falho é iniciado, ele inicia uma eleição

- Se tiver o identificador de processo mais alto que os demais, decidirá que é o coordenador e anunciará isso para os outros processos
- Ele se tornará o coordenador, mesmo que o coordenador corrente esteja funcionando (valentão)

Algoritmo Valentão



A eleição do coordenador p_2, após a falha de p_4 e, depois, de p_3.

Algoritmo Valentão

Esse algoritmo satisfaz à condição de subsistência E2

Não é garantido que o algoritmo satisfaça a condição de segurança E1

- Processos que tenham falhado sejam substituídos por processos com o mesmo identificador
  - Um processo que substitui um processo falho p pode decidir que tem o identificador mais alto
  - Outro processo (que detectou a falha de p) decidiu que possui o identificador mais alto
  - Os dois processos se anunciarão como coordenadores, simultaneamente

Tarefa 6 (parte 2) – postar no Canvas até 28/03/2021

4. No algoritmo valentão, um processo de recuperação inicia uma eleição e se tornará o novo coordenador, caso tenha um identificador mais alto do que o encarregado atual. Essa é uma característica necessária do algoritmo?
5. Sugira como fazer para adaptar o algoritmo valentão para tratar com particionamentos temporários de rede (comunicação lenta) e processos lentos.