Aula 16 - Exclusão mútua distribuída (cont.)

Wednesday, May 4, 2016 13:45

Múltiplos processos acessam 1 RC

- Em um instante de tempo, no máximo 1 processo executa a RC (safety)
- Liveness
- Fairness
- De certa forma, safety e liveness garantem fairness.

Algoritmo de Lamport

- Os processos mantêm um relógio lógico global
- Canais de comunicação FIFO
- Cada processo i mantém uma RQi de requisições ordenadas pelo relógio lógico

Requisição da RC

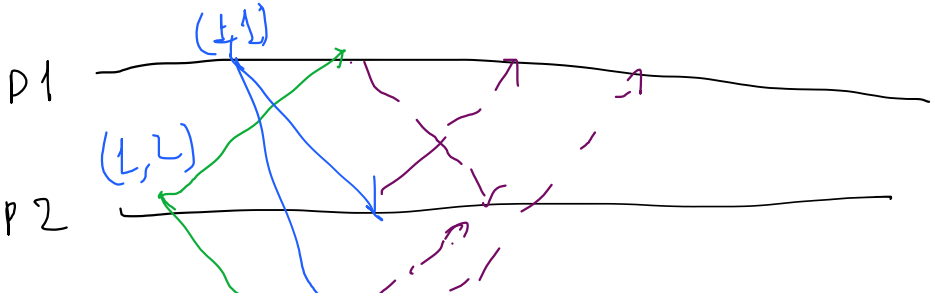
- Para solicitar RC, o processo i envia REQUEST(ti, i) para todos os processos, enfilera a s sua própria requisição na RQi.
- Quando o processo j recebe REQUEST(ti, i), envia um REPLY(tj, j) para o processo i.

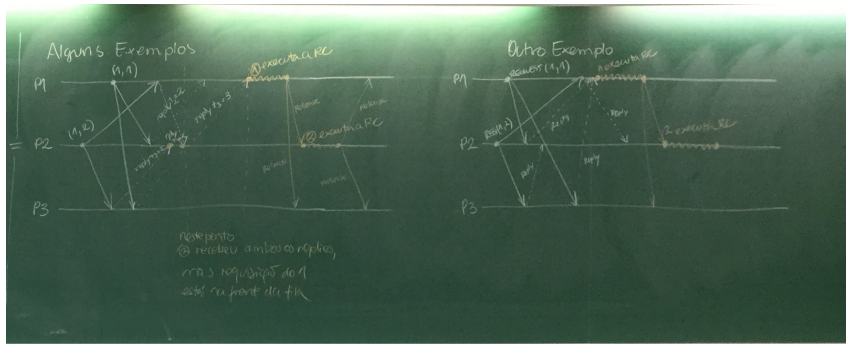
Execução da RC

- Quando as duas condições seguintes L1 e L2 se verificarem, o processo i executa a RC:
  - L1: i recebeu mensagens com timestamp maior que ti de todos os outros proces sos;
  - L2: a requisição de i está na frente de RQi.

Liberação da RC

- O processo i, depois de sair da RC, remove sua requisição da RQi e envia mensagem d e RELEASE para todos os processos.
- O processo j, ao receber o RELEASE, remove a requisição de i de RQj.





Prova de corretude

Teorema 1

O algoritmo de Lamport garante a exclusão mútua

Prova por contradição (absurdo)

Considere que dois processos, i e j, conseguem acesso simultâneo à RC. Para isso acontecer, pela condição L2, cada processo tem sua própria requisição na frente de suas filas.
Sem perda de generalidade, vamos considerar que o timestamp da requisição do processo i é menor que o timestamp da requisição de j.
Ora, quando o processo j executa a RC, a requisição de i só poderia estar na frente da requisição de j em RQj. Portanto: absurdo.

Teorema 2

O algoritmo de exclusão mútua distribuída de Lamport é justo (atende a propriedade de fairness)

Prova

Um algoritmo de exclusão mútua é justo se as requisições são atendidas na ordem definida pelo relógio lógico global.

Mais uma vez, vamos provar por contradição (absurdo).
Considere que um processo i tem requisição com timestamp menor que a requisição de j, **mas** j executa sua requisição antes de i.

Para j conseguir executar a RC, deve ter atendido a condição L1 (j já recebeu mensagens com timestamp maior de todos os outros processos) e a condição L2 (a requisição de i está na frente da sua fila).

Entretanto, a fila é ordenada pelo timestamp das requisições e assim a requisição de i está n a frente da requisição de j.

Avaliação do desempenho

- Para executar a RC, um processo:
  - Transmite (N-1) requests
  - Recebe (N-1) replies
  - Transmite (N-1) releases
  - Total: 3(N-1) mensagens

Uma otimização possível: **Não** enviar reply se já tiver enviado a mensagem com timestamp n maior.
Por exemplo:

- j recebe REQUESTi
- Depois, j envia REQUESTj para i
- j não precisa enviar REPLYi.

No melhor caso, TODOS os replies são evitados! No pior caso, nenhum.
Total de mensagens com a otimização varia de 2(N-1) a 3(N-1).