Exclusão Mútua, Broadcast Causal e Estabilizador de Mensagens: implementação e validação de protocolos*

Alan Teixeira de Oliveira[1] e Pedro Brandão Gonçalves[2]

***Resumo*— Este artigo discute a implementação e validação de algoritmos a partir de três simulações baseadas no protocolo de exclusão mútua, proposto pelos autores Ricart e Agrawala, utilizando relógio lógico escalar e nos protocolos de broadcast causal e do estabilizador de mensagens, propostos por Baldoni e Raynal, usando, respectivamente, o relógio vetorial e o relógio matricial, com o objetivo de validar algoritmos que utilizam relógios lógicos para tratar da comunicação entre processos, a partir da criação de um cenário de sistemas distribuídos.**

I. INTRODUÇÃO

Processo, compreendido como a abstração de um programa em execução, é o conceito central em qualquer sistema operacional (SO) [1]. Cabe ao SO o gerenciamento dos processos, por exemplo, através da alocação de recursos, compartilhamento de dados, troca de informações e sincronização das execuções dos programas [2].

Um processo pode ser compreendido sob três aspectos: contexto de hardware, contexto de software e espaço de endereçamento. O primeiro, contexto hardware, é de grande relevância para os sistemas multiprogramáveis, onde os processos se alternam na utilização do processador, podendo ser interrompidos e, posteriormente, restaurados. O segundo, contexto de software, é o ambiente onde são definidos as identificações, quóruns e privilégios, isto é, especificam-se os limites e as características dos recursos que podem ser utilizados pelos processos. O terceiro, espaço de endereçamento, é a área de memória pertencente ao processo onde as instruções e os dados do programa são armazenados para a execução [2].

Frequentemente, há uma necessidade de comunicação entre os processos, que precisa ocorrer de forma bem estruturada e sem interrupções. Nessa perspectiva, três aspectos devem ser observados: i) como um processo passa informação para outro; ii) como garantir que os processos não entrem em conflito; e iii) como proporcionar uma sequência adequada quando existirem dependências entre processos [1]. Tratar de tais aspectos em sistemas centralizados é mais simples, vez que estes sistemas compartilham memória ou *clock*. Em contrapartida, em sistemas distribuídos, essa situação torna-se mais complexa, visto que os processos não compartilham memória e se comunicam por trocas de mensagens em redes de comunicação [3].

*Trabalho final da disciplina Sistemas Operacionais do Programa de Pós-Graduação em Ciência da Computação - PGCOMP

[1]Bacharel em Ciência da Computação pela Universidade Estadual do Sudoeste da Bahia - UESB, Mestrando do Programa de Pós-Graduação em Ciência da Computação, Universidade Federal da Bahia - UFBA

[2]Bacharel em Sistemas de Informação, pela Universidade Estadual do Sudoeste da Bahia - UESB, Mestrando do Programa de Pós-Graduação em Ciência da Computação,, Universidade Federal da Bahia - UFBA,

No entanto, sabe-se que estabelecer a ordem de precedência causal entre os processos em sistema distribuído é uma forma de ordená-los linearmente pelo momento de sua ocorrência, o que possibilita resolver uma variedade de problemas. Para tanto, acredita-se que o mecanismo de relógios lógicos – escalar, vetorial e matricial – possa ser implementado, visto capturar com precisão essa relação de causalidade entre eventos de processos distintos [4].

Assim, com o objetivo de validar algoritmos que utilizam relógios lógicos, foram realizadas três simulações: a primeira, baseada no protocolo de exclusão mútua, proposto pelos autores Ricart e Agrawala, utilizando relógio escalar; a segunda e a terceira, nos protocolos de broadcast causal e do estabilizador de mensagens, propostos por Baldoni e Raynal, usando, respectivamente, o relógio vetorial e o relógio matricial. Neste trabalho, são apresentados e discutidos os resultados obtidos nessas simulações.

II. PROTOCOLOS INVESTIGADOS

A. Exclusão Mútua

A exclusão mútua é uma forma de assegurar que outros processos sejam impedidos de usar uma variável ou um arquivo compartilhado quando estes já estiverem em uso [1]. Em sistemas distribuídos, por não haver memória compartilhada, não é possível usar objetos compartilhados como semáforos para implementar a exclusão mútua [5]. Entretanto, a literatura dispõe de vários mecanismos para tratar o problema, por exemplo, os relógios lógicos.

Um dos principais mecanismos para tratar o problema da exclusão mútua em sistemas distribuídos é baseado no conceito de relógio lógico escalar propostos por Lamport e o de Ricart e Agrawala. O mecanismo proposto pelos segundos autores é considerado como uma melhoria do primeiro [5][6], e ambos assumem que os processadores e canais de comunicação são confiáveis

Na simulação em análise foi utilizado o algoritmo de Ricart e Agrawala (Figura 1). Este mecanismo usa apenas 2(n-1) mensagens por solicitação da seção crítica. Faz isso combinando as funcionalidades do *ack* (confirmação de mensagens) e *release* (liberação mensagens). Além disso, funciona mesmo quando os canais de comunicação não satisfazem o FIFO (*First-in-first-out*) [5].

B. Broadcast Causal

O broadcast causal é utilizado para reduzir a falta de sincronia dos canais de comunicação à medida que os processos os percebem. Isso significa que as mensagens recebidas por esses processos não podem violar a ordem de precedência

```
Pi::
var
    pending: list of process ids initially null;
    myts: integer initially ∞;
    numOkay: integer initially 0;

request:
    myts := logical_clock;
    send request with myths to all other processes;
    numOkay := 0;

receive (u,request):
    if (u.myts < myths) then
        send okay to process u.p;
    else append(pending, u.p);

receive (u, okay):
    numOkay := numOkay + 1;
    if (numOkay = N - 1) then
        enter_critical_section;

release:
    myths := ∞:
    for j ∈ pendingQ do
        send okay to the process j;
    pendingQ := null;
```

Fig. 1. Protocolo de Exclusão Mútua proposto por Ricart e Agrawala

dos eventos de broadcast correspondentes. Esse conceito tem importância, pois pode ser utilizado para estabelecer uma ordem de prioridades [7].Por exemplo, utilizando o protocolo de transmissão proposto por Baldoni e Raynal (Figura 2) , conforme na simulação em análise, o relógio lógico vetorial associa a cada mensagem um valor, de forma que as mensagens de broadcast, m1 e m2, sejam entregues na sequência, isto é, m1 antes de m2.

```
procedure broadcast(m)
    m.VC:=VCi;
    ∀x ∈ {1,...,n} do send(m) to Px enddo
    VCj[i] := VCj[i] + 1

when Pi receives m from Pj
    delay the delivery until ("x ∈ {1,...,n}:m.VC[x]≤VCi[x]);
    if i ≠ j then VCi[j] := VCi[j]+1;
    deliver m to the upper layer
```

Fig. 2. Protocolo de Broadcast Causal proposto por Baldoni e Raynal

C. Estabilizador de Mensagem

A estabilidade de mensagens é importante para aplicações nas quais os processos transmitem mensagens para todos os outros processos e cada um individualmentedeve receber o mesmo conjunto de mensagens enviadas.

Isso é importante no contexto de transmissões confiáveis, em que cada processo deve armazenar uma cópia da mensagem enviada ou recebida para atender requisitos da aplicação na presença de falhas do processo remetente e partições de rede. Se houver falha, qualquer cópia da mensagem enviada por esse processo poderá encaminhá-la a um terceiro processo. Entretanto, isso pode levar a um crescimento rápido no buffer de cada processo com risco de sobrecarregá-lo. Desta forma, é necessária uma política que reduza a ocorrência de sobrecarregamento de buffer[7].

É notório que não é necessário armazenar no buffer uma mensagem que já foi entregue a todos os destinos pretendidos. O protocolo de detecção de estabilidade de mensagens (Figura 3), proposto por Baldoni e Raynal, fazendo uso do relógio lógico matricial, gerencia os buffers do processo evitando problemas de sobrecarga, conforme poderá se observar na simulação em análise.

```
procedure re1_broadcast(m)
    m.VC:+MCi[i][*];
    m.sender:=i;
    ∀x ∈ {1,...,n} do send(m) to Px enddo
    MCi[i][i] := MCi[i][i] +1

when Pi receives m from Pj
    deposit(m);
    MCi[j][*] := m.VC;
    if i ≠ j then MCi[i][j] := MCi[i][j] + 1;
    deliver m to the upper layer

when(∃ m ∈ buffer_i :m.VC[m.sender] ≤ min_{1≤x≤n}(MCj[x][m.sender]))
    discart(m)
```

Fig. 3. Protocolo Estabilizador de Mensagens proposto por Baldoni e Raynal

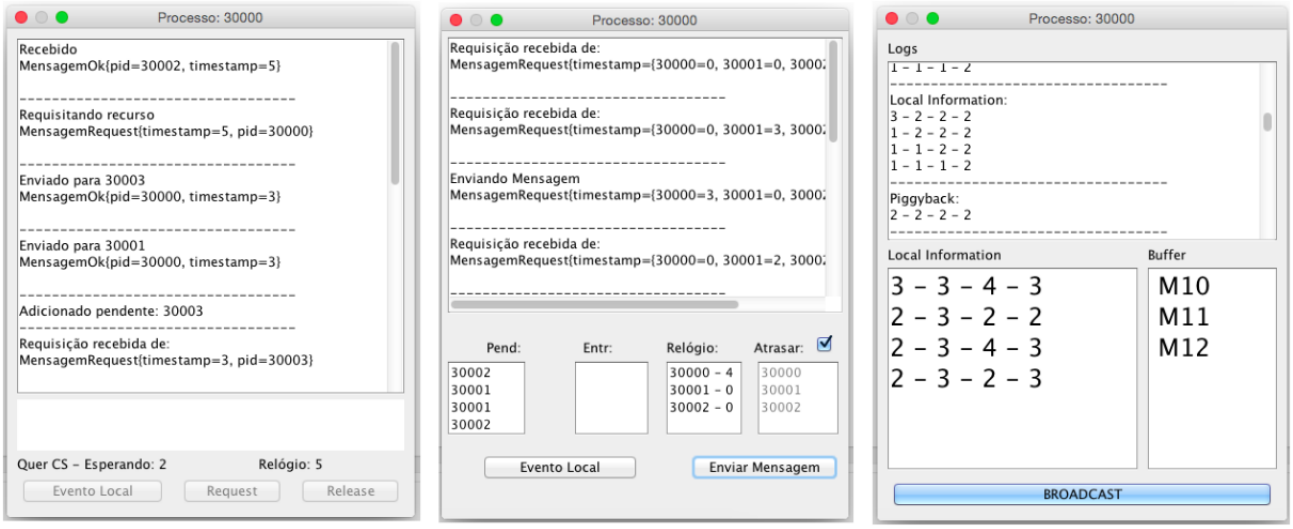
III. METODOLOGIA

Para a simulação dos três protocolos – exclusão mútua, broadcast causal e estabilizador de mensagens – sob investigação, consideramos um modelo sem falhas, isto é, com processadores e canais de comunicação confiável, foi construído um cenário de sistema distribuído em um computador pessoal, utilizando a linguagem de programação JAVA.

A simulação dos processos em diferentes máquinas foi realizada através da implementação de *threads*, e também, do *socket* para tratar da comunicação entre processos.

Utilizando uma linguagem orientada a objetos, o que possibilita a geração de um número de processos escalável, foram implementados os três protocolos. Para visualizar o funcionamento de seus mecanismos, desenvolveu-se uma interface gráfica, a qual dispõe de botões para que se controlem manualmente as ações que os processos podem realizar, conforme demonstra a Figura 4.

Um marcador de atraso foi inserido para as janelas dos processos de broadcast causal, com a finalidade de se verificar a propriedade fundamental do protocolo (preservação da casualidade, já que sem esse mecanismo não seria possível validar tal propriedade, visto que consideramos um modelo sem falha. Os protocolos investigados foram validados, de acordo com as propriedades a seguir e que estão corelacionadas na Tabela I:



(a) Exclusão Mútua (b) Broadcast Causal (c) Estabilizador de Mensagens

Fig. 4. Telas dos protocolos

TABLE I
RELAÇÃO ENTRE PROPRIEDADES E PROTOCOLOS INVESTIGADOS

Propriedades/Protocolos	**Exclusão Mútua**	**Broadcast Causal**	**Estabilizador de Mensagens**
Confiabilidade		x	x
Escalabilidade	x	x	x
Mensagens Estável			x
Ordenação consistente	x	x	x
Preservação da casualidade/Justiça	x	x	
Segurança	x		
Vivacidade	x		

- *Confiabilidade*: a mensagem deve ser recebida por todos os nós em operação.
- *Escalabilidade*: o protocolo se mantém consistente ao passo que se aumenta o número de processos.
- *Mensagens estável*: A mensagem vista por todos os processos deve ser descartada.
- *Ordenação consistente*: mensagens/requisições diferentes enviadas por processos diferentes devem ser entregues a todos os processos na mesma ordem.
- *Preservação da casualidade*: a ordem na qual as mensagens são enviadas deve ser consistente com a casualidade entre os eventos de envio destas mensagens.
- *Segurança*: dois processos não devem ter permissão para usar a seção crítica simultaneamente.
- *Vivacidade*: todo pedido para a seção crítica é eventualmente concedido.

Finalmente, para validar essas propriedades foram criados cenários de testes descritos através de uma tabela para cada protocolo. Estas tabelas apresentam colunas indicando as propriedades, os casos de teste, os resultados esperados e os resultados obtidos, que é a própria validação. A Figura 5 apresenta a tabela para a validação da Eclusão Mútua.

IV. RESULTADOS

A implementação dos protocolos foi feita por meio de pesquisa e desenvolvimento, não sendo, portanto, uma simples tradução direta do pseudocódigo para Java, visto que tais pseudocódigos são disponibilizados com um considerável grau de abstração. Isto pode ser evidenciado no trabalho, uma vez que cada um dos protocolos investigados apresentaram entre 8 a 22 linhas de código e a implementação em JAVA resultou em cerca de 5 classes e 1000 linhas de código por protocolo.

Os protocolos de Exclusão Mútua, Broadcast Causal e do Estabilizador de Mensagens implementados no cenário criado obtiveram todos os resultados esperados, a observação da progressão dos relógios mostrou, de forma prática, como tais mecanismos funcionam.

Na Exclusão Mútua, além dos eventos de envio (*send*) e recebimento (*receive*) de mensagens entre processos, permitiu-se a geração de eventos locais e, ainda assim, o relógio progrediu como deveria. No Broadcast Causal, quando selecionado o marcador de atraso e enviada uma mensagem por um processo, os demais não recebem esta mensagem e, também, não recebem as enviadas posteriormente até que o marcador seja desabilitado e a primeira mensagem seja entregue, isso mostrou que o modelo proposto atende a propriedade da preservação da casualidade. No Estabilizador de Mensagens, pode-se observar o descarregamento do buffer do processo no momento que uma mensagem estável é identificada.

Desta forma, pode-se constatar que os mecanismos de relógios lógicos é um conceito poderoso que pode ajudar programadores, designers e o próprio sistema operacional a resolver uma variedade de problemas em um ambiente distribuído. Este estudo evidenciou que, no designer de algoritmos, a utilização dos relógios lógicos auxilia a garantia de propriedades como a vivacidade e a preservação

Premissas	Casos de teste	Resultados esperados	Validação
Segurança: dois processos não devem ter permissão para usar a seção crítica simultaneamente	Crie 3 processos e faça uma requisição em cada um deles.	Somente um processo tem acesso a seção crítica	✔
Vivacidade: Todo pedido para a seção crítica é eventualmente concedido	Crie 3 processos, faça uma requisição em cada um e depois libere cada um deles.	Todos os processos devem entrar na seção crítica	✔
Justiça : diferentes solicitações devem ser concedidas na ordem em que são feitas.	Crie 3 processos, faça uma requisição em cada um e depois libere cada um deles.	O acesso a seção crítica deve acontecer na ordem em que as requisições foram feitas.	✔
Escalabilidade: o protocolo se mantém consistente ao passo que se aumenta o número de processos.	Execute todos os casos anteriores com 9 processos.	A simulação deve atender os mesmos resultados esperados para cada uma das premissas anteriores.	✔

Fig. 5. Casos de teste para validar a Exclusão Mútua

da causalidade em problemas de exclusão mútua. Também, pode-se observar que em problemas como o de broadcast causal, o conceito de relógios lógicos ajuda na construção de um estado consistente, em muitos casos isso torna possível criar um ponto de verificação para recuperação de falhas e detectar inconsistências em banco de dados replicados. No caso do estabilizador de mensagens, esse conhecimento se mostrou útil para descartar informações desnecessárias, evitar sobrecarregamento de buffers e detectar terminação. Finalmente, saber a ordem de dependência causal dos eventos dependentes ajuda a tratar da simultaneidade na computação, já que todos os eventos que não apresentam uma relação causal podem ser executados ao mesmo tempo.

V. CONCLUSÃO

O artigo abordou a implementação e validação de algoritmos a partir de três simulações baseadas no protocolo de exclusão mútua, proposto pelos autores Ricart e Agrawala, utilizando relógio escalar e nos protocolos de broadcast causal e do estabilizador de mensagens, propostos por Baldoni e Raynal, usando, respectivamente, o relógio vetorial e o relógio matricial. A construção do modelo foi feita na linguagem de programação orientada a objetos JAVA, utilizando *threads* para simular processos e *socket* para tratar da comunicação destes. Posteriormente, foram criados casos de testes para validar o modelo e todos os protocolos implementados obtiveram os resultados esperados, a observação da progressão dos relógios mostrou, de forma prática, como tais mecanismos funcionam.

REFERENCES

[1] A. S. Tanenbaum and N. Machado Filho, *Sistemas operacionais modernos*. Prentice-Hall, 1995, vol. 3.

[2] F. B. Machado and L. P. Maia, *Arquitetura de sistemas operacionais*. LTC, 2004, vol. 4.

[3] A. Silberschatz, P. B. Galvin, and G. Gagne, *Sistemas operacionais: conceitos e aplicações*. Campus, 2001.

[4] M. Raynal and M. Singhal, "Logical time: Capturing causality in distributed systems," *Computer*, vol. 29, no. 2, pp. 49–56, 1996.

[5] V. K. Garg, *Elements of distributed computing*. John Wiley & Sons, 2002.

[6] Y.-I. Chang, "A simulation study on distributed mutual exclusion," *Journal of Parallel and Distributed Computing*, vol. 33, no. 2, pp. 107–121, 1996.

[7] R. Baldoni and M. Klusch, "Fundamentals of distributed computing: A practical tour of vector clock systems," *IEEE Distributed Systems Online*, no. 2, p. null, 2002.