Ambiente de Desenvolvimento e Avaliação de Algoritmos de Exclusão Mútua para Sistemas Distribuídos *

Kêmio de Oliveira Couto† Marco Aurélio de Souza Mendes‡
Osvaldo S. F. Carvalho §

Departamento de Ciência da Computação
Universidade Federal de Minas Gerais
Caixa Postal 702, CEP 30.161-920
Belo Horizonte, MG, Brasil

Resumo

Este trabalho apresenta um ambiente de desenvolvimento, avaliação e comparação, por simulação, de algoritmos de exclusão mútua para sistemas distribuídos. Dois tipos de serviços são fornecidos por esse ambiente: a exportação de um conjunto de primitivas que facilitam a implementação desses algoritmos e ferramentas básicas para análise dos resultados das simulações. O apêndice deste trabalho descreve uma aplicação prática do uso do conjunto de primitivas. Espera-se que este trabalho seja útil para pesquisadores da área de sistemas distribuídos durante a implementação, análise e comparação desses algoritmos.

Abstract

This work presents an environment for development, analysis and comparison, by simulation, of mutual exclusion algorithms in distributed systems. Two services are provided by this environment: the exportation of a set of primitives which facilitates the implementation of these algorithms and basic tools for analysis of the simulation results. The appendix of this work describes a practical application of the use of this set of primtives. It is expected that this work be helpful to researchers of the distributed systems field during the implementation, analysis and comparison of these algorithms.

*Este trabalho foi financiado com recursos da FAPEMIG (TEC 1113/90) e do CNPq (502353/91-O(NV)).
†Aluno de graduação do Departamento de Ciência da Computação da UFMG - e-mail: chalub@dcc.ufmg.br
‡Aluno de pós-graduação do Departamento de Ciência da Computação da UFMG - e-mail: corelio@dcc.ufmg.br
§Professor do Departamento de Ciência da Computação da UFMG - e-mail: vado@dcc.ufmg.br

1 Introdução

A exclusão mútua entre processos num sistema distribuído é uma importante ferramenta para a resolução de muitos problemas de sincronização. Como exemplos clássicos desses problemas, podemos citar o problema dos filósofos à mesa de jantar e problemas do tipo produtor-consumidor. A utilização ou não de algoritmos de exclusão mútua eficientes muitas vezes define o desempenho obtido por um sistema distribuído no gerenciamento de suas tarefas.

Algoritmos distribuídos de exclusão mútua são de difícil depuração e apresentam uma considerável alteração de comportamento com relação às variações de demanda de um sistema. Estas características justificam o desenvolvimento de ferramentas que possibilitem, de maneira confortável, analisar e comparar tais algoritmos.

Este trabalho apresenta um núcleo de descrição de algoritmos distribuídos de exclusão mútua. A função principal desse núcleo de descrição é fornecer uma biblioteca de primitivas, denominada **SiMEA**, que possibilite uma implementação facilitada de algoritmos distribuídos de exclusão mútua, bem como submetê-los a testes por simulação. Complementarmente estão disponíveis ferramentas para análise dos resultados das simulações. O ambiente de desenvolvimento e avaliação de algoritmos de exclusão mútua para sistemas distribuídos (AD) apresentado é formado pelas facilidades oferecidas pelo núcleo de descrição (ND) em conjunto com as ferramentas de análise.

O AD foi desenvolvido sobre a plataforma Unix/XWindow System. Entretanto, a maior parte do código é independente de plataforma. Assim, o AD pode ser adaptado com certa facilidade para outras plataformas.

No decorrer deste artigo, as características do sistema distribuído emulado pelo ND são estabelecidas. O modelo geral de simulação utilizado é descrito. O ND é melhor detalhado, sendo descritas características tais como: conjunto de primitivas, arquivos de configuração e resultados gerados. A seguir, as ferramentas para análise dos resultados são especificadas. Uma análise conclusiva, ao final do artigo, indica possíveis extensões para este trabalho.

2 Histórico

Este artigo é fruto de um trabalho que já vem sendo realizado há algum tempo na área de avaliação e análise de algoritmos distribuídos de exclusão mútua.

Inicialmente algoritmos específicos foram estudados isoladamente, sendo as ferramentas utilizadas nesse estudo adaptadas a cada um deles. No decorrer do trabalho perceberam-se as semelhanças entre as estruturas utilizadas em cada algoritmo, advindo daí a idéia de desenvolver uma ferramenta de propósito geral com o intuito de possibilitar um método padrão para se avaliar quaisquer algoritmos dessa classe.

O projeto dessa ferramenta teve como principais objetivos criar facilidades para a especificação dos algoritmos distribuídos de exclusão mútua, determinar um modelo consistente de simulação e oferecer um suporte estatístico mínimo suficiente para avaliar os resultados das simulações.

3 Características do Sistema

O ND funciona como um emulador de um sistema distribuído. Simulações são realizadas submetendo-se o algoritmo implementado pelo usuário às condições desse sistema. As seguintes características são apresentadas pelo sistema em questão:

- há um único recurso compartilhado;
- os nodos não compartilham memória nem utilizam qualquer relógio comum;
- os nodos se comunicam via troca de mensagens;
- cada nodo pode se comunicar com todos os outros nodos do sistema;
- o tempo de transmissão de mensagens pode sofrer atrasos aleatórios, mas a transmissão é livre de erros;
- mensagens de um nodo i para um nodo j são entregues na ordem de envio;

- cada nodo do sistema assegura a serialização de sua demanda interna, isto é, no máximo uma demanda está ativa por vez e o recurso é liberado antes que o nodo passe a processar a próxima demanda;
- o algoritmo em questão é executado igualmente por todos os nodos, sem qualquer falha;
- as rotinas do algoritmo são executadas de maneira local e indivisível.

4 Modelo de Simulação

A fim de que os resultados obtidos na simulação de um algoritmo sejam mais confiáveis, faz-se necessário a criação de um modelo que implemente e permita a variação dos diversos parâmetros do sistema em estudo. O modelo aqui detalhado é baseado no proposto em [DHP].

A cada nodo do sistema são associados três estados distintos:

Thinking: Estado associado ao período de latência do nodo.

Hungry: Estado associado ao período de solicitação do recurso compartilhado pelo nodo.

Eating: Estado associado ao período de utilização do recurso compartilhado pelo nodo.

O modelo utilizado associa a cada nodo i do sistema três parâmetros d_i, U_i e R_i, que são o tempo de transmissão de mensagens, o tempo de utilização do recurso compartilhado e o tempo de latência ou descanso, respectivamente. Os parâmetros do sistema d_i, U_i e R_i são variáveis aleatórias e distribuídas estatisticamente.

Especificados os parâmetros anteriores para um dado algoritmo de exclusão mútua A, podemos definir para cada nodo as funções $\mathcal{M}_i(d, U, R, A)$ e $\mathcal{W}_i(d, U, R, A)$, onde a primeira determina o número de mensagens enviadas por demanda e a segunda o tempo de espera para utilização do recurso compartilhado. Os parâmetros e as funções estão sintetizados na tabela abaixo.

d_i	Tempo de transmissão de mensagens
R_i	Tempo de latência
U_i	Tempo de uso do recurso compartilhado
$\mathcal{M}_i$	Número de mensagens por demanda
$\mathcal{W}_i$	Tempo de espera

5 Núcleo de Descrição

A utilização do ND apresenta duas fases bastante distintas. Na primeira fase, o usuário deve codificar o algoritmo a ser avaliado num arquivo em linguagem C++ e *ligá-lo* com a biblioteca de primitivas disponíveis **SiMEA**, gerando um arquivo executável. Na segunda fase, o usuário deve criar um arquivo que contenha a parametrização necessária à configuração do sistema distribuído a ser simulado. Concluídas essas fases, o programa gerado pode ser executado, acessando o arquivo de configuração e gerando como resultado arquivos de dados que servirão para análise e amostragem do comportamento do algoritmo.

5.1 Linguagem Escolhida

A linguagem C++ foi escolhida para o desenvolvimento deste trabalho, por ser:

- uma linguagem orientada por objetos;
- apresentar uma grande difusão, principalmente no meio acadêmico.

Grande parte do trabalho proposto neste artigo se baseia em simulações. O uso da linguagem C++ se justifica pelo fato de simulações serem as aplicações que mais diretamente se beneficiam das técnicas orientadas por objetos [Meyer].

A linguagem C tornou-se em poucos anos uma das mais populares linguagens de programação. A linguagem C++, por ser uma extensão orientada por objetos da linguagem C, acrescenta várias facilidades e apresenta total compatibilidade com qualquer programa C.

5.2 Conjunto de Primitivas

A biblioteca **SiMEA** exporta um conjunto de primitivas (classes, funções, variáveis, constantes) para a especificação de algoritmos distribuídos de exclusão mútua. Essas primitivas oferecem subsídios para a representação dos principais elementos formadores de um sistema distribuído.

Do ponto de vista dos algoritmos distribuídos de exclusão mútua, os nodos e as mensagens são os elementos principais, constituindo os objetos a serem representados computacionalmente. A partir do exposto, são exportadas as seguintes classes:

NODE: É a representação de um nodo de um sistema distribuído;

TMESSAGE: É a representação de uma mensagem.

Essas classes servem de modelo para a construção de classes especializadas através do mecanismo de herança. As classes especializadas devem representar o nodo e a mensagem do sistema distribuído em estudo.

Um nodo do sistema em questão deve reagir a três eventos: solicitação do recurso, liberação do recurso e recebimento de mensagens. A classe **NODE** deixa disponível funções virtuais que visam exatamente tratar esses eventos, sendo elas **HUNGRY**, **THINK** e **MESSAGE**, respectivamente.

A especificação de um algoritmo utilizando o ND consiste no processo de dar corpo às funções acima, e prover uma função construtora para a devida inicialização de variáveis.

Dois serviços complementares são oferecidos pela biblioteca **SiMEA** para utilização pelos nodos do sistema:

Send: Primitiva para troca de mensagens entre nodos;

Eat: Função que provoca a entrada de um nodo em sua região crítica.

Dado que o nodo do sistema distribuído já foi *descrito*, surge a necessidade de criar o sistema propriamente dito através do instanciamento dos nodos que o compõem. Com esse intuito, a biblioteca **SiMEA** exporta uma variável denominada **PointersToNodes**. Essa variável é um vetor, onde cada posição deverá conter o endereço de um nodo do sistema.

Acima foram vistas as funções que efetivamente participam do processo de especificação de um algoritmo. Para a execução das simulações estão disponíveis mais duas funções, são elas:

Config: Processa o arquivo de configuração de simulação;

Simulator: Responsável por todo o processo de simulação.

Outras primitivas que complementam a biblioteca **SiMEA** e detalhes de utilização são descritos no Apêndice A.

5.3 Arquivo de Configuração de Simulação

O arquivo de configuração de simulação (.simrc) deve ser fornecido pelo usuário antes da execução da simulação. Tal arquivo informa ao ND os parâmetros sobre os quais a simulação será executada. Os seguintes parâmetros devem ser definidos:

- número de nodos do sistema;
- tempo médio de latência de cada nodo;
- tempo médio de utilização do recurso de cada nodo;
- tempo médio de transmissão de mensagens entre um nodo e os demais nodos do sistema;
- desvio padrão associado a cada tempo acima;
- distribuições estatísticas;
- tempo de simulação.

Tal arquivo possui uma sintaxe própria para a configuração dos parâmetros de simulação. Essa sintaxe proporciona de uma maneira flexível a definição de todos os parâmetros necessários, como pode ser visto no exemplo e melhor detalhado no Apêndice B.

```
// Arquivo de configuracao de um sistema distribuido com 7 nodos
NODES 7
// configuracao de cada nodo
NODE  1 1000, 100, 1000, 100, 200, 20, EXPONENTIAL, EXPONENTIAL
NODE  2  500,  40, 1000, 100, 200, 20, EXPONENTIAL, EXPONENTIAL
NODE  3  500,  40, 1000, 100, 200, 20, EXPONENTIAL, EXPONENTIAL
NODE  4  500,  40, 1000, 100, 200, 20, EXPONENTIAL, EXPONENTIAL
NODE  5  500,  30, 1000, 100, 200, 20, EXPONENTIAL, EXPONENTIAL
NODE  6  300,  30, 2000, 200, 200, 20, EXPONENTIAL, EXPONENTIAL
NODE  7  300,  30, 2000, 200, 200, 20, EXPONENTIAL, EXPONENTIAL
// definicao da distribuicao para o tempo de envio de mensagens
D_DISTRIBUTION UNIFORM
// definicao do tempo de simulacao
TIME 1000
// finaliza arquivo
END
```

5.4 Resultados Gerados Pelo Núcleo de Descrição

Normalmente, o principal fator levado em consideração na determinação da eficiência de um algoritmo distribuído de exclusão mútua é o número de mensagens por demanda. O tempo de espera para a obtenção do recurso comum é outro fator relevante. Informações auxiliares, como tempo de ciclo e carga sobre o recurso compartilhado, podem ser trivialmente derivadas a partir dos resultados acima citados. Baseado nisso, a cada execução do ND são dados como resultados o número médio $\mathcal{M}$ de mensagens enviadas por demanda de exclusão mútua, que é calculado como a média dos $\mathcal{M}_i$, e o tempo médio de espera $\mathcal{W}$ para a utilização do recurso, que é calculado como a média dos $\mathcal{W}_i$.

Cada execução do ND gera dois arquivos (*results.sim* e *states.sim*). O primeiro exibe os valores de $\mathcal{M}$ e $\mathcal{W}$. O segundo exibe as mudanças de estado dos nodos durante a simulação, sendo formado por duas colunas; a primeira contém o número do nodo e a segunda o número referente ao estado para o qual esse nodo transiciona (THINKING = 1, HUNGRY = 2 e EATING = 3). O exemplo desses arquivos foi obtido com a execução do algoritmo de [Ricart-Agrawala]:

results.sim

```
12.00     31.95
```

states.sim

```
7   2
6   2
3   2
2   2
4   2
5   2
6   3
1   2
6   1
7   3
7   1
2   3
2   1
3   3
3   1
4   3
4   1
5   3
5   1
1   3
1   1
```

O algoritmo especificado pelo usúario através do ND pode eventualmente conter algum erro lógico. A detecção de erros leva ao término da execução do ND, invalidando logicamente os resultados que seriam gerados. Durante a execução, o ND detecta a ocorrência de dois tipos de erros:

Violação da Exclusão Mútua: Ocorre quando mais de um nodo utiliza, ao mesmo tempo, o recurso compartilhado;

Ocorrência de Deadlock: É determinada pela formação de ciclos, isto é, um grupo de nodos não consegue acessar o recurso compartilhado.

6 Ferramentas de Análise

As ferramentas de análise dos resultados de simulações oferecem ao usuário meios de avaliar o comportamento de um algoritmo distribuído de exclusão mútua. As ferramentas disponíveis são duas:

Visualização Gráfica: Este programa realiza várias simulações sobre o algoritmo implementado e constrói gráficos em duas dimensões com base nos resultados presentes no arquivo *results.sim*.

Acompanhamento de Estados: Este programa trabalha sobre o resultado de uma única simulação. O seu objetivo é mostrar a dinâmica de alternação de estados de um nodo determinada pelo algoritmo implementado. Atua sobre as informações presentes no arquivo *states.sim*.

6.1 Visualização Gráfica

Esta ferramenta permite avaliar as principais características dos algoritmos a serem implementados, através da geração de gráficos que refletem o comportamento dos mesmos. A geração de gráficos para dois algoritmos distintos possibilita a comparação indireta entre eles.

Os dados gerados a cada simulação são os valores de $\mathcal{M}$ e $\mathcal{W}$ que, combinados com as variações em R e U, possibilitam a confecção de quatro tipos de gráficos:

W x R: Gráfico do tempo de espera em função do tempo de latência;

M x R: Gráfico do número de mensagens em função do tempo de latência;

W x U: Gráfico do tempo de espera em função do tempo de uso do recurso compartilhado;

M x U: Gráfico do número de mensagens em função do tempo de uso do recurso compartilhado.

A fim de obter uma maior representatividade, o valor de d é usado como unidade, i.e., os valores de R, U e $\mathcal{W}$ apresentados nos gráficos são na realidade R/d, U/d e $\mathcal{W}/d$. Um complemento de grande importância estatística é o cálculo do intervalo de confiança, sendo que a confiança é um dos parâmetros fornecidos pelo usuário.

Os arquivos gerados são nomeados WxR, MxR, WxU e MxU e apresentam formato texto. Permitindo o uso de várias interfaces gráficas para visualização efetiva.

Esta ferramenta chama-se **makegraph** e trabalha sobre os seguintes argumentos de chamada:

Protocolo: Especifica o nome do arquivo executável que servirá como alvo das simulações;

R/U: Determina quais parâmetros de simulação serão variados: R_i ou U_i;

Repetição: Informa quantas vezes uma simulação será repetida com os parâmetros de simulação invariáveis. Quanto maior este valor mais lenta será a execução, mas menor será o erro (intervalo de confiança) alcançado;

Limite superior: Define o valor máximo até onde os valores dos R_i ou U_i serão variados, isto é, o maior valor no eixo X. Um valor elevado para este argumento possibilita avaliar casos limites onde $R \rightarrow \infty$ ou $U \rightarrow \infty$.

Passo: Especifica o intervalo de variação dos parâmetros R/U. Quanto menor este intervalo mais lenta será a execução, mas o gráfico obtido provavelmente aproximará melhor a verdadeira função.

Confiança: Determina qual será a confiança desejada. Um valor entre 0 e 1 é esperado.

6.2 Acompanhamento de Estados

O objetivo desta ferramenta é oferecer uma visão simplificada de como ocorrem as transições de estado dos nodos do sistema quando da simulação de um algoritmo. Como visto anteriormente, um nodo pode apresentar três estados: THINKING, HUNGRY e EATING. A representação gráfica desses estados procura ser a mais intuitiva possível, como visto abaixo.

Esta ferramenta foi implementa sobre XWindow System e é denominada **xstates**. O único argumento requerido por esse programa é o nome do arquivo executável sobre simulação.

O valor desta ferramenta é basicamente didático, não fornecendo muitas informações sobre características do algoritmo simulado.

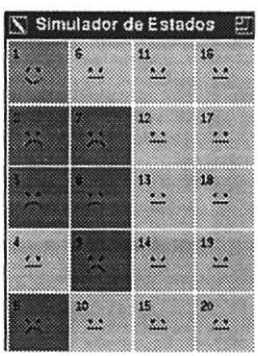

Figura 1: Fotografia do Acompanhamento de Estados

7 Conclusões

Este trabalho apresentou um ambiente de desenvolvimento e avaliação de algoritmos de exclusão mútua para sistemas distribuídos, cujo objetivo é permitir desenvolver, analisar e comparar esses algoritmos de forma simples e confortável. Espera-se que o AD seja utilizado como ferramenta de apoio por pesquisadores da área de sistemas distribuídos, e em aulas práticas de uma disciplina que aborde esse tipo de algoritmos.

O AD vem sendo utilizado há algum tempo, com os mais importantes algoritmos distribuídos de exclusão mútua já estando implementados sobre ele. A implementação e teste de dois algoritmos citados na literatura usando esse ambiente já foi alvo de estudo, gerando um artigo submetido ao XIII Concurso de Trabalhos de Iniciação Científica.

Durante a utilização do AD percebeu-se a importância desse ambiente visto que, a partir das primitivas e facilidades fornecidas por ele, os esforços puderam ser concentrados somente em tarefas relacionadas diretamente ao processo de implementação e simulação dos algoritmos propriamente ditos. Com isso, os algoritmos podem ser implementados em um curto período de tempo e uma metodologia de testes e análise é conseguida, dando mais consistência a qualquer estudo feito nesta área.

Um complemento natural para este trabalho, já em estudo, seria a extensão para lidar com outros problemas de exclusão mútua, tal como exclusão mútua com N recursos compartilhados, por exemplo. Uma outra possibilidade é o desenvolvimento de um ambiente de depuração. Nesse ambiente o usuário poderia interferir nos eventos que se sucederiam ao longo da simulação, por exemplo, a determinação do momento da chegada de uma mensagem ou a especificação do instante em que cada nodo entrará em estado de solicitação do recurso. Essa interferência ajudaria na detecção de erros, ao mesmo tempo que possibilitaria ao usuário forçar situações extremas que colocariam em teste a robustez do algoritmo em estudo.

Apêndice A

Classe TMESSAGE

Classe básica de uma mensagem.

- Protótipo:

```
class TMESSAGE {
public:
        int message;
        TMESSAGE (int m);
        virtual ~TMESSAGE() {};
};
```

- Atributos:

message: Identificador do tipo de mensagem.

- Métodos:

Construtor TMESSAGE: Associa um identificador à mensagem quando do instanciamento de um objeto desta classe.

Classe NODE

Classe básica que representa um nodo de um sistema distribuído.

- Protótipo

```
class NODE {
protected:
        int  Number;
        char State;
public:
        int Id();
        void Change_State(char new_state);
        NODE(int n);
        virtual void HUNGRY() = 0;
        virtual void MESSAGE(int From, TMESSAGE* Message) = 0;
        virtual void THINK() = 0;
};
```

- Atributos:

Number: Identificador do nodo;

State: Identificador do estado deste nodo (THINKING, HUNGRY ou EATING).

- Métodos:

Construtor NODE: Associa um número como identificador do nodo e inicializa seu estado para THINKING;

obs: As funções seguintes formam o cerne do processo de desenvolvimento dos algoritmos, portanto devem ser obrigatoriamente implementadas pelo usuário.

Construtor da classe herdeira de NODE: No construtor desta classe todo o processamento necessário a inicialização das variáveis deve ser realizado;

THINK(): Esta função é executada quando um nodo passa para o estado de descanso (EATING → THINKING);

HUNGRY(): Esta função é executada quando da transição do estado de latência para o estado de solicitação do recurso (THINKING → HUNGRY);

MESSAGE(int From, TMESSAGE* Message): Esta função é chamada todas as vezes em que é detectada uma mensagem cujo destinatário é o nodo.

obs: Os demais métodos dessa classe não precisam ser utilizados quando da construção de um algoritmo. Tais métodos são acessados internamente pelo ND.

Função Send

- Protótipo:

```
void Send (int From, int To, TMESSAGE* Message);
```

- Uso: Esta função envia uma mensagem de nodo From para o nodo To.

Função Eat

- Protótipo:

```
void Eat(int Node);
```

- Uso: Esta função faz com que um determinado nodo entre em sua região crítica.

Função Config

- Protótipo:

```
int Config( void );
```

- Uso: esta função é responsável pela leitura do arquivo de configuração de simulação (.simrc). Seu valor de retorno determina o número total de nodos do sistema. Tal número é definido no arquivo .simrc pelo comando **NODES num_nodes**.

Função Simulator

- Protótipo:

```
void Simulator( void );
```

- Uso: Esta função faz a execução da simulação propriamente dita, realizando o controle do tráfego de mensagens e das mudanças de estado dos nodos.

Variável PointersToNodes

Esta variável deve ser inicializada pelo usuário. Ela é um vetor de apontadores para objetos do tipo **NODE**. Esse vetor deverá conter os endereços dos nodos do sistema. Esses nodos serão instanciados pelo usuário a partir de uma classe especializada de **NODE**.

Constantes

O estado associado a cada nodo pode assumir o valor de uma das três constantes abaixo. Sendo que o usuário pode comparar o estado corrente com esses valores para processar algum tratamento específico para um dado estado.

_THINKING

_HUNGRY

_EATING

A criação da função principal **main()** deve ser feita pelo usuário. Nessa função deve-se acessar o arquivo de configuração de simulação, instanciar os nodos do sistema e finalmente executar a rotina de simulação propriamente dita. Para exemplificar, esta é a rotina principal do algoritmo de [Carvalho-Roucairol]:

```
main()
{
        int Number_of_Nodes = Config( );
        for (int i=0; i < Number_of_Nodes ;++i)
                PointersToNodes[i] = new ROUCAIROL(i+1, Number_of_Nodes);
        Simulator( );
        return 0;
}
```

Apêndice B

O arquivo de configuração de simulação pode ser visto como uma sequência de comandos finalizados por um comando END. Abaixo são descritos os comandos disponíveis:

//: Comentário. Ignora qualquer caracter até o final da linha.

NODES: Define o número de nodos do sistema. Espera como parâmetro um inteiro positivo.

NODE: Define os tempos médios e desvios padrões associados a cada nodo assim como as distribuições associadas.

Possui a seguinte sintaxe:

NODE num R, SR, U, SU, d, Sd, R_Distribution, U_Distribution

O parâmetro **num** define o número do nodo ao qual serão associados os demais parâmetros. Os parâmetros R, U e d definem, respectivamente, o tempo médio de latência, o tempo médio de utilização do recurso e o tempo médio de transmissão de mensagens entre um nodo e os demais nodos do sistema. Os parâmetros SR, SU e Sd definem, respectivamente, os desvios padrões associados a R, U e d. Por último, os parâmetros R_Distribution e U_Distribution definem a distribuição estatística para o tempo de latência e para o tempo de utilização do recurso, respectivamente.

D_DISTRIBUTION: Define a distribuição estatística para o tempo de transmissão de mensagens. Espera um parâmetro que defina qual será a distribuição utilizada.

TIME: Define o tempo de simulação. Espera um parâmetro inteiro positivo que defina esse tempo.

END: Finaliza o arquivo de configuração de simulação.

Para especificar as distribuições estatistísticas a serem utilizadas os seguintes identificadores são disponíveis:

UNIFORM: Distribuição uniforme;

RANDOM: Distribuição randômica;

EXPONENTIAL: Distribuição Exponencial;

HYPER_EXPONENTIAL: Distribuição Hiper-exponencial;

ERLANG: Distribuição de Erlang;

NORMAL: Distribuição Normal.

Apêndice C

Este é um algoritmo simples baseado na implementação dada em [Raynal] para exemplificação do uso do conjunto de primitivas da biblioteca **SiMEA**.

Algoritmo de Ricart-Agrawala

```
**************************AGRAWALA.H************************************

#ifndef AGRAWALA_H
#define AGRAWALA_H

#include "simulator.h"

#define NIL     0
#define REQ     1
#define REP     2

class AGRAWALA : public NODE  {
        int n;
        long osn;
        long hsn;
        int numrepexpect;
        char csrequested;
        char priority;
        char* repdeferred;
        int Precedes(long CLOCKi, long CLOCKj, int i, int j);
public:
        AGRAWALA(int , int);
        void HUNGRY();
        void MESSAGE(int, TMESSAGE*);
        void THINK();
};
#endif
```

```
**************************AGRAWALA.C************************************

#define max(a,b) (a>b) ? a:b
#include "Agrawala.h"

class MMESSAGE : public TMESSAGE {
public:
        long TimeStamp;
        MMESSAGE(int tm, long ts) : TMESSAGE(tm) { TimeStamp = ts; }
}

AGRAWALA::AGRAWALA (int Num, int Number_of_Nodes) : NODE(Num) {
        n = Number_of_Nodes; repdeferred = new char[n];
        hsn = 0; csrequested = priority = 0;
        for(int i=0; i < n ;i++) repdeferred[i] = 0;
}

int AGRAWALA::Precedes(long CLOCKi, long CLOCKj, int i, int j) {
        return ( (CLOCKi < CLOCKj) || ( (CLOCKi == CLOCKj)  &&  (i < j) ) );
}

void AGRAWALA::HUNGRY() {
        csrequested = 1; osn = hsn + 1; numrepexpect = n-1;
        for (int node=1; node <= n ;node++)
                if ( node != Number ) Send(Number, node, new MMESSAGE(REQ, osn));
}

void AGRAWALA::MESSAGE(int From, TMESSAGE* Message) {
        MMESSAGE* Mesg = (MMESSAGE*) Message;
        switch(Mesg->message) {
        case REP:
                if ( !(--numrepexpect) ) Eat(Number);
                break;
        case REQ:
                hsn = max(hsn, Mesg->TimeStamp);
                priority = csrequested && Precedes(osn, Mesg->TimeStamp, Number, From);
                if ( priority ) repdeferred[From-1] = 1;
                else Send(Number, From, new MMESSAGE(REP, hsn));
                break;
        }
}

void AGRAWALA::THINK(void) {
        csrequested = 0;
        for (int i=0 ; i < n ; ++i)
                if (repdeferred[i])
                   { repdeferred[i] = 0; Send (Number, i+1, new MMESSAGE(REP, hsn)); }
}

main()
{
        int Number_of_Nodes = Config( );
        for (int i=0; i < Number_of_Nodes ;++i)
                PointersToNodes[i] = new AGRAWALA(i+1, Number_of_Nodes);
        Simulator( );
        return 0;
}
```

Referências

[Lamport] Lamport, L. *Time, clocks, and ordering of events in a distributed system.* Commun. ACM 21,7 (July, 1978), 558-565.

[Chandy-Misra] Chandy, K.M., and Misra J. *The drinking philosophers.* ACM Trans. Program. Lang. Syst. 6,4 (Oct. 1984), 632-646.

[Dijkstra65] Dijkstra, E.W. *Co-operating sequential processes.* in Programming Languages, Genuys, F. (ed.), Academic Press, London, 1965

[Dijkstra75] Dijkstra, E.W. *Guarded commands, nondeterminacy and formal derivation of programs.* Commun. ACM 18,8 (Aug. 1975), 453–457.

[Raynal] Raynal, M. *Algorithms for mutual exclusion.* ISBN 0-262-18119-3

[Ricart-Agrawala] Ricart, G., and Agrawala, A.K. *An Optimal algorithm for mutual exclusion in computer networks.* Commun. ACM 24,1 (Jan. 1981) 9-17.

[Carvalho-Roucairol] Carvalho, O.S.F., and Roucairol,G. *On mutual exclusion in computer networks.* Commun. ACM 26,2 (Feb. 1983), 146-147.

[Carvalho-Campos] Carvalho, Osvaldo S.F. and Campos, Sérgio V.A. *A* $0..\sqrt{n}$ *distributed mutual exclusion algorithm.*

[Maekawa] Maekawa, Mamoru. *A* $\sqrt{N}$ *algorithm for mutual exclusion in decentralized systems.* ACM trans. Comp. Syst. 3,2 (May 1985), 145-159.

[DHP] Dupuis, Alan and Hebuterne, Gérard and Pitie, Jean-Marc. *A Comparison of Two Mutual Exclusion Algorithms for Computer Noteworks.* Note Technique CNET/LAA/SLÇ 1985.

[CMC93] Couto, K.O., Mendes, M.A.S., e Carvalho, O.S.F. *Uma Comparação Entre Dois Algoritmos de Exclusão Mútua para Redes de Computadores.* ANAIS do V SBAC-PAD, Volume I, 358-367

[LEX] Lesk, M.E., and Schmidit, E. LEX - *A Lexical Analyser Generator.* Bell Laboratories – Murray Hill, New Jersey 07974.

[Meyer] Meyer, Bertrand. *Object-Oriented Software Construction.* ISBN 0-13-629049-3

[Stroustrup] Stroustrup, Bjarne. *The C++ programming language.* ISBN 0-201-53992-6