Escalonador de Tarefas de Tempo Real Para um Simulador de Sistemas Distribuídos

Jaqueline M Suzart *
Instituto Federal de Educação, Ciência e Tecnologia da Bahia
Avenida Araújo Pinho, 39 - Canela, Salvador - BA, 40110-090
jaqueline.suzart@ifba.edu.br

Allan Edgard Freitas †
Instituto Federal de Educação, Ciência e Tecnologia da Bahia
Avenida Araújo Pinho, 39 - Canela, Salvador - BA, 40110-090
allan@ifba.edu.br

ABSTRACT
Developing a distributed system is a complex task, and it is difficult to assure system works fine after development has been done. That may require the use of verification and validation, which can be performed by using simulators. A distributed system simulator characterizes the behavior of components and attributes of a distributed system so that it can execute by simulating its environment, matching different possibilities of component's behaviors for that system. That should include real-time behavior. With the development of a real- time scheduling module for a distributed system simulator it´s possible to simulate periodic tasks and different schedules, and to characterize the processing according to the information given and if this task will get priority or not, setting the next steps to follow

RESUMO
Para o desenvolvimento de um sistema distribuído é necessário um conhecimento mais específico devido a sua complexidade, e mesmo tendo o conhecimento, isso não garante que o sistema depois de pronto funcionará corretamente. Por isso, atualmente são utilizados simuladores para que o projeto pensado seja verificado e validado. O simulador de sistemas distribuídos caracteriza o comportamento dos componentes do sistema e seus atributos para execução do ambiente de simulação, combinando diferentes possibilidades de comportamento dos componentes e isto inclui o comportamento de tempo real. O simulador de escalonamento de tempo real desenvolvido permite a simulação de tarefas periódicas de tempo real e tipos de escalonamento, sendo possível a caracterização do processamento de acordo com as informações cadastradas e se esta tarefa terá ou não prioridade, determinando assim os próximos passos a serem realizado.

Palavras - chaves
Escalonador, Tempo Real, Sistemas distribuídos, EDF, RM, Simulador.

---

* Graduanda do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas

† Doutor em Ciência da Computação (Orientador)

1. INTRODUÇÃO

Um sistema computacional pode, sob supervisão do sistema operacional, executar de forma concorrente um conjunto de processos (i.e. programas em execução). A execução de uma rotina de escalonamento permite definir o acesso ao processador para a execução efetiva de cada processo, observadas as prioridades de execução, quando houver, e a política de escalonamento a ser utilizada.

No desenvolvimento dos sistemas computacionais pode ser necessário o uso de técnicas de avaliação de desempenho, como a simulação computacional, em que, ao invés da execução de um sistema completamente implementado, parte do comportamento do sistema é executada em um ambiente controlado de simulação [1].

A avaliação dos sistemas computacionais por simulação deve também simular o comportamento o escalonamento de processos, garantindo assim a veracidade da simulação dos sistemas.

Em especial, num cenário onde as tarefas possuem requisitos temporais mais restritos, ditas de tempo real, deve-se observar que o escalonamento de tarefas de tempo real exige observância pelos algoritmos das restrições inerentes aos limites temporais das tarefas [2].

Um escalonador de tempo real deve observar as peculiaridades de cada tarefa, desde as que são periodicamente ativadas (como rotinas de verificação de um sistema, por exemplo), ditas tarefas periódicas, até tarefas de ativação eventual, ditas aperiódicas. Tarefas com requisitos de tempo real devem executar dentro de uma janela temporal restrita, sem que haja a perda do tempo máximo de execução, conhecido como *deadline*, uma vez que a execução dentro do limite temporal é parte do requisito de corretude da solução proposta. Exemplos desses sistemas vão desde simples fornos de microondas até sistemas de controle automático de voo, como, por exemplo, o controle de pouso de um avião o qual deve acionar o trem de pouso tão logo seja detectada a proximidade do solo através de um sensor de altitude – a execução correta desta ação depende necessariamente do atendimento do limite temporal imposto pelo *deadline* [2]. Para que isso seja garantido algumas premissas devem ser cumpridas.

O tempo de ativação e o *deadline* das tarefas são uns dos principais fatores para a definição das premissas, pois é através da observância destes que será possível ao algoritmo garantir o comportamento correto da aplicação em um sistema de tempo real (STR). Além disso, para desenvolver tarefas de sistemas de tempo real é necessário o conhecimento do desempenho das tarefas e garantir o atendimento das restrições temporais.

Em cenários como o de sistemas distribuídos, diferentes nós computacionais podem executar tarefas de tempo real e interagirem entre si por meio de troca de mensagens.

O estudo destes cenários por meio de avaliação de desempenho pode ser efetuado por diferentes técnicas, como, por exemplo, medição, modelagem analítica e simulação [1]. A avaliação de desempenho por meio de simulação de sistemas distribuídos com restrições de tempo real deve combinar aspectos de simulação de sistemas distribuídos e aspectos de simulação do escalonamento de tarefas de tempo real.

Nesta proposta de trabalho, apresentamos um simulador de escalonamento de tempo real a ser integrado a um simulador de sistemas distribuídos para tarefas periódicas e pré definidas, sendo possível realizar o escalonamento por meio de dois tipos de algoritmos de escalonamento diferentes e largamente utilizados na literatura: o EDF e o RM.

O presente trabalho está organizado em seções descritas abaixo:

- Seção 2: descreve a definição e importância dos sistemas distribuídos e aspectos de tolerância a falhas nestes;
- Seção 3: define os sistemas de tempo real;
- Seção 4: define aspectos de escalonamento de tempo real, descrevendo alguns dos principais algoritmos utilizados;
- Seção 5: descreve aspectos de avaliações de desempenho de sistemas computacionais;
- Seção 6: discute de forma breve alguns trabalhos correlatos;
- Seção 7: apresenta a ferramenta proposta, sua importância e finalidade;
- Seção 8: apresenta a conclusão e os trabalhos futuros.

2. SISTEMAS DISTRIBUÍDOS E ASPECTOS DE TOLERÂNCIAAFALHAS EM SISTEMAS DISTRIBUÍDOS

Sistemas distribuídos são caracterizados por um conjunto de processos espalhados em diversos nós computadores conectados por meio de uma rede de computadores e que se comunicam entre si por meio de troca de mensagens [3]. Desta forma, os sistemas distribuídos são compostos por processos e canais de comunicação que permitem a troca de mensagens entre tais processos.

Por exemplo, a Figura 1 apresenta um sistema distribuído composto por 3 processos (programas em execução), P1, P2 e P3, os quais se comunicam entre si por meio dos canais de comunicação indicados na Figura – observe que P2 e P3 estão hospedados em um mesmo nó computador e P1 está em um nó distinto. Os canais de comunicação abstraem detalhes da infraestrutura subjacente: sockets, sistema operacional, rede, links de comunicação etc.

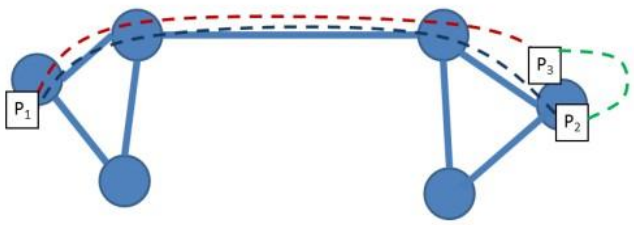

Figura 1: Exemplo de um sistema distribuído [4].

Sistemas distribuídos se apresentam com diferentes requisitos temporais. Em sistemas ditos síncronos, há limites temporais conhecidos para a execução de tarefas e para a comunicação entre processos. Por outro lado, o sistema é considerado assíncrono quando não é possível fazer suposições sobre a velocidade de execução de processos, nem sobre os atrasos na entrega de mensagem [3].

Nos sistemas ditos assíncronos, os componentes ou nós distribuídos executam passos e trocam mensagens entre si sem limites temporais previamente conhecidos, sendo um modelo mais genérico de interação que se adéqua para representar e desenvolver qualquer sistema. [3]

Considere, por exemplo, a comunicação entre os sítios distintos pela Internet, esta comunicação é assíncrona: caso haja instabilidade na rede, o nó que enviou a mensagem não tem como saber que o outro nó que não recebeu a mensagem falhou ou esta mensagem foi perdida, ou ainda está a caminho, em face da ausência de limites temporais.

Na comunicação síncrona, existe um limite máximo e mínimo para execução e resposta do processo, já conhecido, onde cada processo tem um relógio local com taxa conhecida e limitada no desvio, com relação ao tempo real. Em caso de uma resposta a uma mensagem não ser recebida em uma janela de tempo pré-determinado, é possível inferir uma falha no nó computacional envolvido com a computação [3].

Ainda, há sistemas que combinam as características síncronas e assíncronas, denominados como sistemas distribuídos parcialmente síncronos, em que a existência de limites temporais pode variar de acordo com a disponibilidade de recursos, em configurações híbridas (i.e. diferentes canais de comunicação podem apresentar diferentes características, ou síncronas ou assíncronas), ou dinâmicas (i.e. na presença de recursos necessários um canal de comunicação pode se apresentar síncrono, e ao longo da execução, perder tais características e tornar-se assíncrono) [4].

2.1 ASPECTOS DE TOLERÂNCIAA FALHAS

Os componentes de um sistema distribuído (processos e canais de comunicação) estão sujeitos a falhas. Diferentes modelos de falhas definem os tipos de falhas que ocorrem com maior probabilidade nos sistemas distribuídos. A falha afeta o funcionamento esperado de um componente do sistema, isto pode se refletir em um erro na informação provida por este componente para o sistema e este erro em um defeito do sistema, ou seja, o sistema se comporta de forma não esperada [5].

A grande maioria dos defeitos encontrados são causados por falhas de componente físico, pois são as principais causas do problema. A falha dos nodos de rede e a falha de comunicação são exemplos de problemas físicos e podem ser classificados, segundo Cristian [6], nas categorias de omissão, colapso, temporização, de resposta e arbitrária. A falha por omissão é quando um componente não responde a uma solicitação. A falha de colapso ou *crash* ocorre quando a partir de um dado momento, o componente pode falhar por omissão e continua falho (i.e. sem responder) até ser reiniciado, se isto for possível. A falha de temporização acontece quando um componente responde a solicitação, mas o resultado enviado está fora do intervalo de tempo definido. A falha de resposta também responde a solicitação, mas o resultado enviado não está correto. A falha arbitrária ou bizantina é uma falha genérica, englobando também todas as possibilidades já descritas, de modo que o componente pode se comportar de forma totalmente inesperável e imprevisível [6].

Uma das principais vantagens de sistemas distribuídos é a possibilidade de se manter aplicações tolerantes a falhas, por meio de mecanismos como o da replicação de processos, garantido a continuidade do serviço em execução, mesmo que ocorram falhas em um determinado número de processos e canais de comunicação [7].

3. SISTEMAS DE TEMPO REAL

Atualmente, muitos sistemas têm requisitos que demandam comportamento de tempo real, como, por exemplo, o sistema de frenagem ABS do carro, em que o comando deve ter o resultado da atuação em uma janela de tempo predefinida, tornado cada vez mais comuns aplicações desenvolvidas em sistema de tempo real (STR). O Sistema de Tempo Real (STR) é um sistema computacional que deve reagir a estímulos oriundos do seu ambiente em prazos específicos [2].

O STR pode ser classificado como sistemas críticos e brandos. STR críticos possuem premissas de tempo restritas, as quais se não forem cumpridas podem causar consequências graves, como por exemplo, uma catástrofe [2].

Um sistema é considero brando quando o não cumprimento temporal causa pouco impacto, e não impede a continuação do funcionamento do sistema e nem causa consequências graves.

Um sistema STR é considerado um sistema reativo, pois o sistema ou parte dele interagem diretamente com o ambiente enviando respostas para as entradas vindas do mesmo [2].

No STR os prazos e tempos determinados (*timeliness*) para execução devem se cumpridos para assim garantir o comportamento correto da aplicação STR.

Além da propriedade, a determinação da previsibilidade do comportamento do sistema é outro critério para a garantia do cumprimento dos prazos estipulados para execução da tarefa. Previsibilidade é a capacidade do sistema em executar ações desejadas dentro de intervalos de tempos determinados a priori e, via de regra, ditados pelo próprio ambiente no qual o sistema computacional está imerso [2].

Para que seja garantida a execução da tarefa no seu *timeliness*, deve-se observar parâmetros de execução como *deadline*, período e tempo máximo de execução [2].

Deadline é o tempo máximo que uma tarefa tem para ser executada. Período é o tempo entre duas ativações de uma mesma tarefa dita periódica. O tempo máximo das ativações consecutivas de uma tarefa também impõe um tempo máximo para a execução da tarefa, *deadline* [2]. Tempo de computação (*Worst Case Computation Time*) é o tempo total necessário para a execução de uma tarefa, se não houvesse preempção.

Uma tarefa é considerada como periódica quando se sabe o exato momento em que a tarefa será ativada (que a cada período de tempo ocorre nova ativação) e aperiódica é quando não é conhecido um período de ativação.

Na Figura 2, é representada a ativação de uma tarefa periódica onde é definida diante das restrições temporais citadas temos então o comportamento temporal de uma tarefa periódica Ti descrito pela tripla (Ci, Di, Pi). Onde Ci representa o tempo de computação da tarefa, Pi é o período da tarefa, e Di é o "deadline". Nessa representação de tarefas periódicas, Di é uma grandeza relativa (intervalo), medida a partir do início do período Pi. O "deadline" absoluto e o tempo de liberação da k-ésima ativação da tarefa periódica Ti são determinados a partir dos períodos anteriores [8].

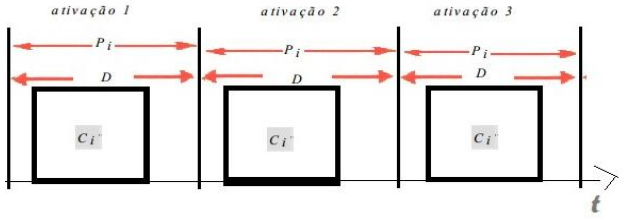

Figura 2: Ativação de tarefa periódica

Numa tarefa aperiódica, o tempo de computação e o *deadline* são os parâmetros principais e cruciais para execução. Na tarefa (Ci, Di, mini), que é representada na Figura 3, é apresentada com duas requisições. Tomando o tempo de chegada da requisição esporádica 2 como a2, o "*deadline*" absoluto desta ativação assume o valor dado por: d2=a2+Di [5].

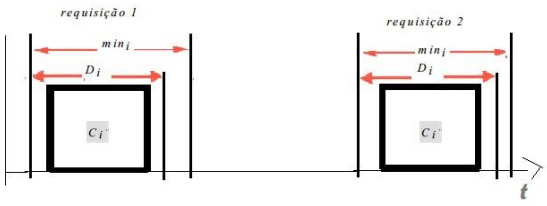

Figura 3: Ativação de tarefa aperiódica

Para executar todas as tarefas do STR e cumprir as restrições definidas, um componente crucial é o escalonamento de tarefas, o qual deve observar as restrições e requisitos do sistema desenvolvido, garantido assim a eficácia da execução.

4. ESCALONAMENTO

O escalonador é um módulo de sistema computacional que escalona a execução de um conjunto de tarefas, onde as premissas que definiram os critérios de execução variam de acordo com o algoritmo selecionado.

Um evento ocorre quando há uma mudança no estado da tarefa, quando uma tarefa chega ao estado de atendimento, é colocada em uma fila de execução. A tarefa que é selecionada na fila de execução passa para o estado executando. O escalonador é o módulo do sistema operacional que faz essa seleção [8]. A Figura 4 apresenta o diagrama de estados para a execução de uma tarefa.

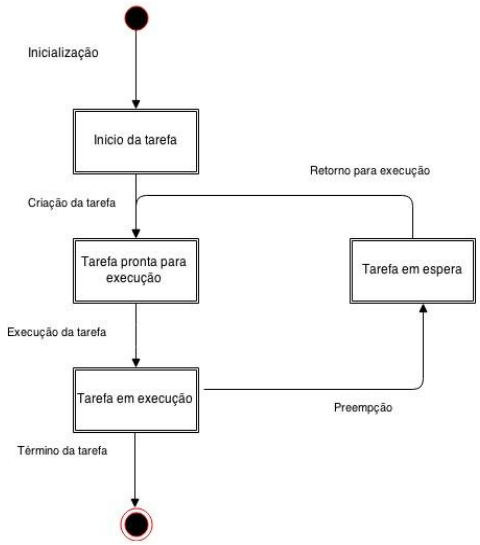

Figura 4: Diagrama de estado de execução da tarefa

Ao selecionar uma tarefa para execução, o sistema operacional verifica se o processador está em uso, e caso esteja, a tarefa que está no processador é retirado; por fim, a tarefa que foi selecionada na fila de execução será colocada no processador.

Esse processo de retirada de uma tarefa do processador para adição do outro processo é denominado de troca de contexto. Ao realizar o escalonamento de uma tarefa pode-se associar um quantum de tempo alocado para a execução da mesma.

A tarefa é executada até que o quantum de tempo alocado para a execução termine, ou até que a tarefa bloqueie em face de requisitar acesso a um recurso, ou até que sofra preempção de uma tarefa de maior prioridade para execução (caso de uma interrupção, por exemplo). O intervalo de tempo de cada tarefa é calculado como uma função da prioridade do processo quando há uma liberação do processador [9].

Para definir qual processo passará para a fila de execução e será executado, são utilizados critérios como prioridade ou período. Esses critérios serão selecionados de acordo com o algoritmo de escalonamento definido.

Alguns dos algoritmos de escalonamento mais utilizados para escalonar em sistemas de tempo real são o *Rate Monotonic* (RM) e o *Earliest Deadline First* (EDF).

4.1 *Rate Monotonic* (RM)

No algoritmo *Rate Monotonic* (RM) a ordem de execução é definida por prioridade e baseado no período (intervalo em que uma tarefa solicita ser escalonada) das tarefas, sendo que quanto menor o período, maior sua prioridade no conjunto de tarefas [10].

As premissas utilizadas pelo RM são:

- As tarefas são sempre periódicas e independentes;
- O *deadline* de cada tarefa é igual ao seu período (Di = Pi); e.
- O tempo de computação de cada tarefa é constante e conhecido.
- O tempo de chaveamento entre tarefas é assumido como nulo.

Para atender as premissas temporais para a execução de *n* tarefas, as condições da fórmula apresentada na Equação 1 devem ser satisfeitas, onde C_i é o tempo de computação da tarefa i e P_i, o seu período:

$$U = \sum_{i=0}^{n} \frac{Ci}{Pi} \leq n\left(\sqrt[n]{2} - 1\right)$$

Equação 1: Verificação se uma tarefa é escalonável pelo RM

A prioridade de cada tarefa é estática e diretamente proporcional a sua ativação, considerando o $T_i = D_i$, ou seja, a execução de cada tarefa deve finalizar antes da ativação da próxima tarefa [11].

Quando uma tarefa não satisfaz as condições das premissas do RM, a tarefa pode não ser executada corretamente, e ocorrer à perda do *deadline* como é apresentado no exemplo da Figura 5, onde, são apresentadas duas tarefas A e B possui período de 5 unidades de tempo e 2 unidades de tempo de computação, e A possui período de 7 e tempo de computação de 4 unidades de tempo.

Neste exemplo U = 2/5+4/7 = 0,97, o que é superior a 2(√2-1) ~ 0,69, ou seja, não há garantia que o escalonamento atenderá os requisitos temporais, o que se verifica na Figura 5 com um time overflow, ou perda de *deadline*.

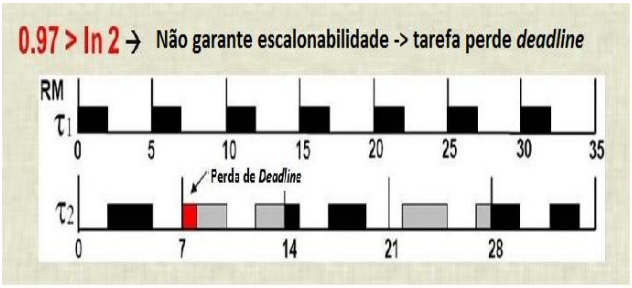

Figura 5: Exemplo de perda de *deadline* em escalonamento com RM.

À medida que a quantidade de tarefas cresce, a verificação apresentada na Equação 1 converge para uma utilização do processador de 0,69. Uma utilização de aproximadamente 70% define uma baixa ocupação do processador que, certamente, implica no descarte de muitos conjuntos de tarefas com utilização maior e que, mesmo assim, apresentem escalas realizáveis [8].

4.2 *Earliest Deadline First* (EDF)

O *Earliest Deadline First* (EDF) é um algoritmo de escalonamento baseado em prioridade dinâmica, a política de escalonamento do EDF corresponde a uma atribuição dinâmica de prioridades que define a ordenação das tarefas segundo os seus "*deadlines*" absolutos (di) [5].

As premissas utilizadas pelo EDF são [8]:
- As tarefas são sempre periódicas e independentes;
- O "*deadline*" de cada tarefa coincide com o seu período (Di=Pi);
- O tempo de computação (Ci) de cada tarefa é conhecido e constante;
- O tempo de chaveamento entre tarefas é assumido como nulo.

No EDF, a escalonabilidade é verificada em tempo de projeto, tomando como base a utilização do processador. Um conjunto de tarefas periódicas satisfazendo as premissas acima é escalonável com o EDF se e somente se atender ao disposto na Equação 2 [8].

$$U = \sum_{i=1}^{n} Ci/pi \leq 1$$

Equação 2: Verificação se uma tarefa é escalonável pelo EDF

Como o comportamento do algoritmo EDF é similar ao do RM, será realizado o somatório para a verificação se os fatores de utilização (Ui) das tarefas do conjunto não ultrapasse o número de processadores disponíveis para suas execuções [5]. Porém, para o EDF, a utilização possível do processador é de 100%, logo o resultado do somatório das tarefas que serão escalonadas, para garantir a sua execução sem perda de *deadline*, não podem ser maior que 1.

5. AVALIAÇÃO DE DESEMPENHO

A avaliação de desempenho de um sistema é um dos fatores principais para a construção do modelo, isso porque na avaliação de desempenho que se determina o tempo de resposta e a desempenho dos sistemas. Um ponto de partida para a avaliação de um sistema é conceber um modelo do mesmo.

Um modelo representa o comportamento de uma realidade ou parte dela. Este conceito é geral. Portanto, aplica-se a qualquer área do conhecimento, onde os modelos sejam necessários [12].

A construção do modelo pode ser através da modelagem do sistema, utilizando qualquer informação possível e necessária para o desenvolvimento do sistema.

Para coletar essas informações, devem-se observar as seguintes atividades [12]:

I. Estudar e interpretar a realidade a ser modelada: deve- se conhecer a realidade a ser modelada e extrair os pontos importantes, ou seja, os reais significados para a construção do modelo desejado;

II. Listar os pontos a serem avaliados e identificá-los no modelo: garante a convergência da aplicação das técnicas, impedindo a utilização inadequada do modelo;

III. Escolher a ferramenta mais apropriada para a tarefa de modelagem: uma avaliação criteriosa possibilita a escolha da ferramenta com as características mais adequadas ao modelo;

IV. Construir e verificar a validade do modelo: Após construir o modelo, deve-se validá-lo, verificando se os resultados previstos pelo mesmo são confimados no sistema real. Caso o sistema real não exista ou não esteja disponível, pode-se validar o modelo através de comparação dos resultados de duas técnicas diferentes;

V. Garantir a confiabilidade dos resultados: é importante que para cada resultado obtido, seja conhecida a margem de erro associada.

Para a realização da modelagem, é necessária a definição de qual tipo de modelo será utilizado para o desenvolvimento do sistema. Os tipos de modelos que podem ser utilizados são: físico, analítico e simulação.

Um modelo físico são construções, normalmente em escala, que permitem a avaliação e definição de algumas características do sistema real. Modelo analítico é construído através de equações matemáticas e diagramas. Além dos diagramas, pode ser utilizada uma descrição que será executada em alguma ferramenta computacional e por fim os de simulação são expressões de comportamento em linguagem algorítmica [12].

Para a construção a definição de qual modelo será realizada a modelagem, é necessário definir qual técnica será utilizada. Estas técnicas utilizam-se de ferramentas de previsão, que possibilitam a modelagem dos sistemas segundo algum paradigma. Os resultados fornecidos por estas técnicas estarão tão próximo da realidade quanto á precisão do modelo empregado, isso não significa um modelo detalhado e complexo, mas um modelo que seja suficientemente preciso para fornecer resultados que possibilitem decisões corretas [12].

As técnicas de avaliação de desempenho utilizadas variam de acordo com o modelo utilizado. As técnicas analítica são utilizadas para modelos analíticos e as de simulação para modelos funcionais, sendo essa relação apresentada no Quadro 1 [12].

	Métodos Elementares	**Métodos Indiretos**	
		Simulação	Analíticos
Sujeito da Avaliação	Sistema real	Método Funcional	Método Computacional
Nível de Abstração	Nenhum	Baixo	Alto
Fator limitante	Tempo de observação	Tempo de simulação	Complex. do Algoritmo
Precisão	Real	Alta	Extra

Quadro 1: Relação entre técnicas e modelos [12].

Das técnicas utilizadas para a avaliação de desempenho, podemos destacar as técnicas de aferição, de modelagem analítica e de simulação.

Na técnica de aferição é realizada a construção de protótipos, coletas de dados e *benchmarks*. A construção de protótipos é uma implementação simplificada do sistema real e com abstração das características essenciais. A coleta de dados é realizada através da observação das atividades de um sistema coletando as características relevantes para a realização da análise do mesmo. No *benchmarks* é utilizado um ponto fixo ou referência para que seja feita a análise e as comparações para um padrão preestabelecido [13].

Na técnica de modelagem analítica, são utilizadas ferramentas matemáticas e computacionais para a implementação, como redes de Petri, *Statecharts* e rede de filas.

A rede de Petri é uma técnica de modelagem que permite a representação de sistemas, utilizando como alicerce uma forte base matemática [14, 15]. O *Statecharts* é usado para descrever sistemas através dos estados de transições e comportamentos (dentro os

estados ao longo das transições) [16]. A rede de filas é um ramo da probabilidade que estuda o fenômeno da formação de filas de solicitantes de serviços, que são providos por um determinado recurso [13].

Para que seja possível mapear de forma correta como o sistema irá se comportar, é utilizada a simulação, onde ela deve refletir todas as informações e características do sistema, para que seja possível garantir o resultado obtido com a simulação.

Uma simulação combina a execução algorítmica existente na aferição com modelos representados por modelagem analítica [7]. De fato, a simulação de um sistema computacional executa as características e o comportamento de cada componente do sistema computacional como um modelo algorítmico que executa o modelo analítico associado ao componente.

Um exemplo de avaliação de desempenho por simulação é a avaliação de um ambiente que faz escalonamento de processos [13].

Através da avaliação é possível analisar:

I. Adequação de um índice de carga;
II. Utilização de diferentes arquiteturas;
III. Utilização de diferentes políticas de escalonamento

5.1 Simulação de Sistemas

A simulação de sistemas pode ser definida como a construção de um programa computacional para implementar modelos de fenômenos ou sistemas dinâmicos e o modelo pode ser uma representação válida do sistema em estudo [13].

As vantagens para utilização da simulação são: versatilidade, pois é aplicada em diferentes situações, flexibilidade, já que é adaptável a novas situações, baixo custo, porque com um mesmo programa pode-se simular diferentes situações do mesmo problema e a facilidade de manipulação [13].

Para a elaboração e validação da simulação, são necessárias algumas condições como: a validação do modelo, elaboração e teste de programas, poucas restrições aos modelos. O resultado de uma simulação é probabilístico, quando é simulado as condições de entrada por meio de funções estatísticas – é esperado que em rodadas distintas de execução o sistema não receba a mesma entrada.

A simulação de sistemas distribuídos é efetuada utilizando ferramentas que mapeiam em que recurso o erro pode ocorrer, como por exemplo, em processadores, rede de comunicação, relógios, memória não volátil e no próprio software.

Um exemplo de utilização de simuladores é a investigação do comportamento de protocolos de maneira controlada, sendo que a execução da mesma simulação pode ser repetida diferentes vezes para uma melhor análise estatística dos resultados. Com o uso de simuladores, não é preciso ter disponível o sistema real onde o protocolo será executado [7].

A implementação correta de protocolos deve satisfazer as propriedades de suas especificações para diversos cenários de defeitos e interações com o ambiente. Devido à complexidade dos sistemas distribuídos, é difícil assegurar que propriedades não sejam violadas como resultado de erros de especificação de um projeto ou implementação. Métodos como verificação formal e modelagem analítica têm sucesso na modelagem do comportamento de subsistemas simples, mas em casos complexo o uso de simulações detalhadas permite obter de forma razoável o comportamento esperado [17].

6. TRABALHOS CORRELATOS

A área de simulação tem produzido ao longo dos anos um grande número de ferramentas que permite modelar sistemas distribuídos e de rede [4], possibilitando a modelagem e análise de sistemas computacionais de redes de computadores e sistemas distribuídos, mesmo não tendo a disponibilidade do sistema real.

Podemos citar algumas ferramentas que realização essa função como, por exemplo: Spin, CESIUM, SMPL, NS-2, OMNET++, Neko, Simmcast e HDSS.

Spin é uma ferramenta para análise de consistência lógica de protocolos e algoritmos distribuídos, sendo composto basicamente por duas partes, já que é responsável pela verificação formal exaustiva das propriedades do algoritmo, procurando identificar a existência de *deadlocks*, por exemplo, e a outra parte executa a simulação. Porém a linguagem em que foi desenvolvido, Promela, não permite que a arquitetura de um protocolo seja especificada diretamente e não possui protocolos já implementados e suporte para redes típicas da Internet como NS [19]. A Figura 6 apresenta a estrutura do Spin.

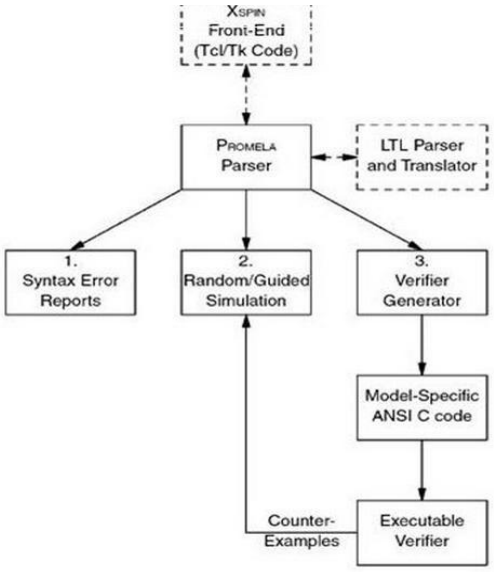

Figura 6: Estrutura do SPIN [19].

O CESIUM (*Centralized Distributed-Eecution Simulator with Failure Modeling*) é um ambiente orientado a objetos para teste de implementação de protocolos distribuídos tolerantes falhas. Ele permite a execução de protocolo incluído à ocorrência de defeitos e ataques de segurança sobre condições controladas, incluindo a ocorrência de defeitos e ataques de segurança durante a execução [20].

O SMPL (*Simple Potable Simulation Language)* é uma extensão funcional da linguagem C, que as operações de simulação são executadas através de chamadas das funções do subsistema de simulação [18]. Uma visão geral do SMPL é apresentada na Figura 7.

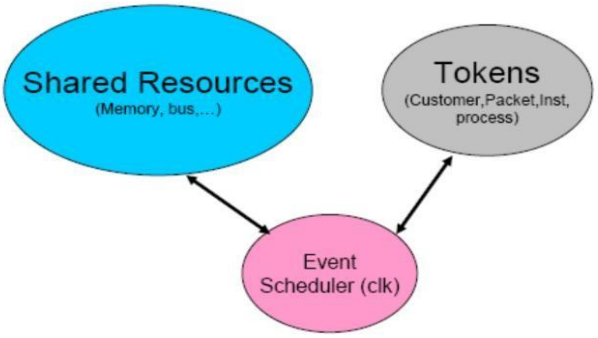

Figura 7: Visão dos sistemas da SMPL [18].

O NS2 é um simulador de eventos para redes de comunicação que realiza a simulação para diferentes tipos de rede/protocolo, provendo simulação de protocolos que acessam topologias pré definidas a partir de scripts. Nele é possível realizar o estudo de políticas de filas, controle de congestionamento, roteamento, desempenho de protocolos, dentre outros. Um exemplo de utilização seria a perda em enlaces, simulando o modelo de falhas de omissão em canais de comunicação [19].

OMNET++ é um simulador baseado em eventos discretos que combina C++ e a linguagem de alto nível denominada NED. Ele permite a simulação de redes, porém não permite a simulação de redes distribuída baseada em comportamento fim-a-fim [21].

Neko e Simmcast são ferramentas de simulação que também conseguem modelar sistema distribuído em que o foco é na comunicação fim-a-fim. Eles também permitem a reutilização de código para uso em protótipo em uma rede de computadores reais e são baseados em *frameworks* de simulação de eventos discretos em Java [4].

O Neko trabalha com modelo de falha por *crash*, podendo ser estendido para outros modelos de falhas, porém as suas primitivas de agendamento de tarefas não garantem a execução em uma janela de tempo estrita [22].

O Simmcast trabalha com conceito de relógio global de eventos, permitindo o agendamento de eventos temporizados para cada *thread*, simulando de acordo com a linha do tempo global, porém o simulador não prevê a simulação de relógio físico local para cada processo [20].

O HDDSS (*Hybrid and Dynamic Distributed System Simulator*) é um framework que realiza simulação de sistemas distribuídos híbridos e dinâmicos com comunicação síncrona ou assíncrona, com troca de mensagens por meio de sockets UDP em uma rede de computadores real e variações de sistemas parcialmente síncronos.

O HDDSS foi desenvolvido permitindo simular sistemas distribuídos que possuem componentes cujo comportamento pode variar ao longo do tempo, de acordo, por exemplo, com a disponibilidade de recursos ou a ocorrência de falhas [4].

O HDDSS é considerado como um ambiente de simulação baseado em um motor de execução serial de eventos discretos, dito núcleo de execução, orientado por eventos e pelo tempo (tarefas podem ser agendadas na ocorrência de eventos ou pelo tempo), onde é capaz de simular sistemas distribuídos híbridos e dinâmicos, bem como distribuídos convencionais, como síncronos, assíncronos e parcialmente síncronos [4].

Na Figura 8 é apresentada uma visão simplificada dos componentes de uma simulação. O componente principal do núcleo de execução é a classe Simulator, a qual executa uma instância de simulação de acordo com um arquivo de configuração. O arquivo de configuração define a configuração inicial de processos e canais de comunicação e a dinâmica do sistema [4].

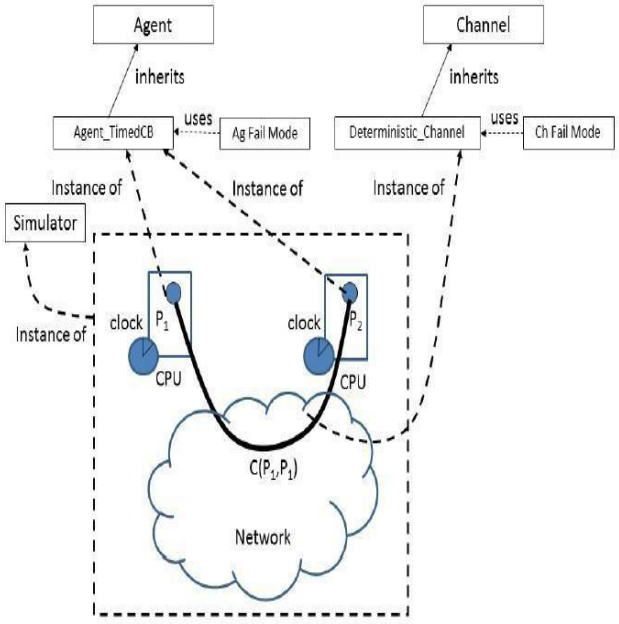

Figura 8: Visão simplificada dos componentes do simulador HDSS

Além dos simuladores de sistemas distribuídos, têm-se também os simuladores para sistemas de tempo real, podendo demonstrar que a não escalonabilidade apontada na simulação restringe-se a uma perda de *deadline* que causa um prejuízo insignificante ao sistema, além de apontar quando esta perda realmente acontecerá [24]. Para isso pode-se citar o STRESS, RTSIM e Simulador de Escalonamento para Sistemas de Tempo Real.

STRESS é um ambiente geral para análise e simulação do comportamento de aplicações de tempo real, tendo como objetivo a avaliação do algoritmo de escalonamento e administração de recurso. É composto de uma linguagem genérica para descrição de arquitetura, kernel e aplicação; recursos de análise para kernel e aplicações; e recursos de simulação, além de exibir graficamente o escalonamento simulado [23].

RTSIM (*Real Time System Simulator*) é uma ferramenta para definição, simulação e análise de sistemas de tempo real, que contempla de simples cenários com poucos tipos de tarefas num único nó até várias arquiteturas complexas constituídas de múltiplos nós conectados com uma rede e compartilhando um conjunto de recursos, sendo baseada em uma biblioteca de simulação de eventos discretos chamada *MetaSi* [24].

O Simulador de Escalonamento para Sistemas de Tempo Real é uma ferramenta de comparação de eficiência dos algoritmos de escalonamento, que é efetuada de acordo com a execução de um sistema composto por tarefas e pelo algoritmo a ser avaliado [25].

7. PROPOSTA

A proposta desse trabalho é desenvolver um módulo escalonador de tempo real para uso no simulador de sistema distribuído HDDSS.

O HDDSS foi escolhido por ser um simulador de sistemas distribuídos que caracteriza o comportamento dos componentes do sistema e seus atributos para execução do ambiente de simulação, combinando diferentes possibilidades de comportamento dos componentes do sistema.

O escalonamento efetuado e desenvolvido para o *framework* de simulação HDDSS será executado pelo usuário onde deverá

informar as premissas necessárias para o escalonamento de processos periódicos. Além de cadastrar processos, o usuário poderá visualizar os processos em andamento como mostra o diagrama de caso de uso abaixo.

No sistema deve ser possível cadastrar e escalonar tarefas com ou sem prioridade, sendo elas informadas pelo usuário. Quando inicializado o escalonamento, o sistema verificará se a política de escalonamento é EDF ou RM e logo após, verificará se todas as tarefas cadastradas são escalonáveis, caso não, ela será retirada da fila de processo de escalonamento. O fluxo do sistema é representado pelo diagrama da Figura 9.

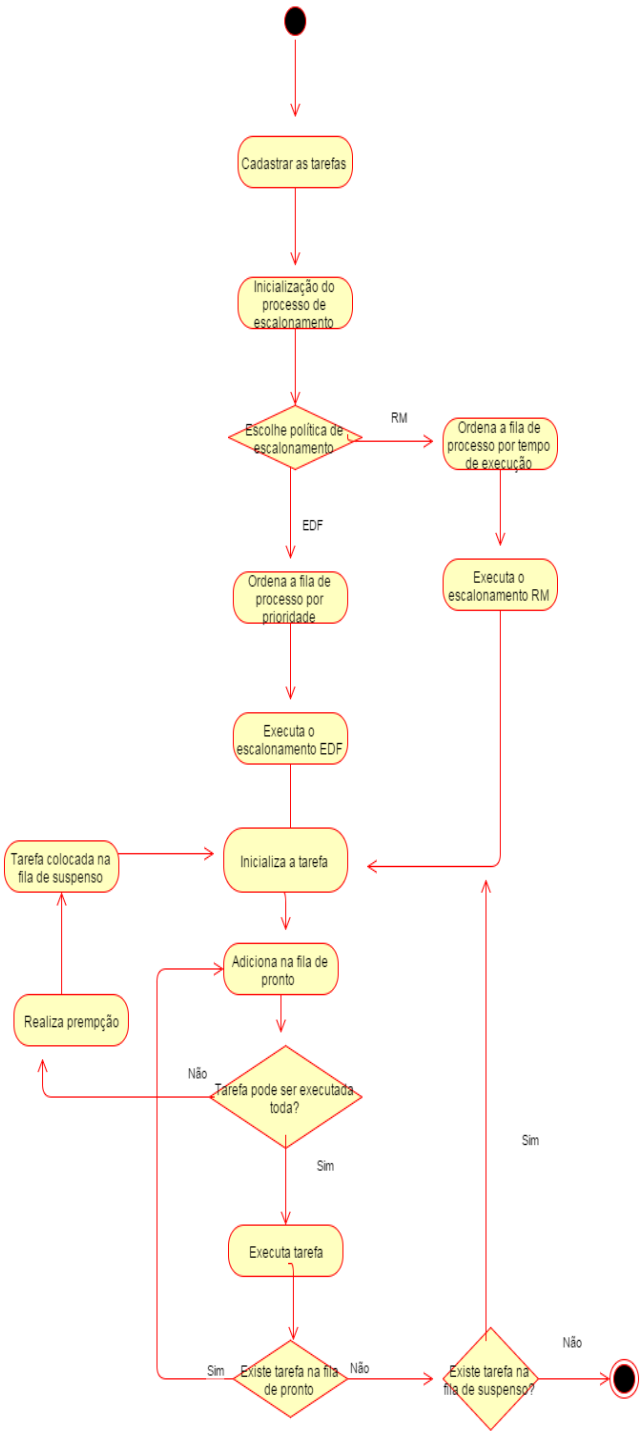

Figura 9: Diagrama de atividade

Uma interface foi desenvolvida para operar diretamente com o módulo de simulação de escalonamento. Ao acessar esta interface é exibida uma tela de apresentação e através do menu, o usuário escolhe a opção para cadastrar as tarefas que irão ser escalonadas. As Figuras 10, 11 e 12 ilustram as telas da interface de uso.

O sistema apresenta a tela de cadastro dos processos e nela é possível informar os dados do processo como o nome, a periodicidade do processo, seu tempo de execução, quantidades de vezes que o processo será escalonado e a prioridade de cada elemento que será escalonado, caso tenha.

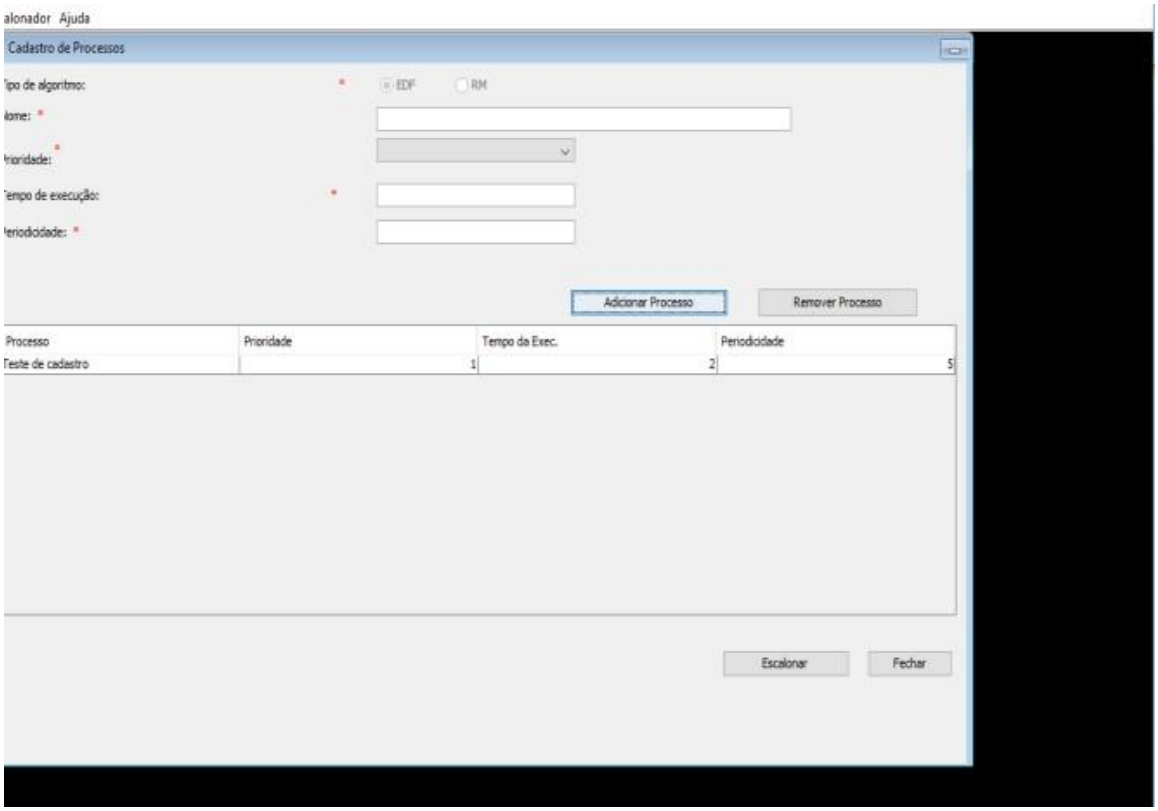

Figura 10: Tela de cadastro de tarefas para o escalonamento EDF.

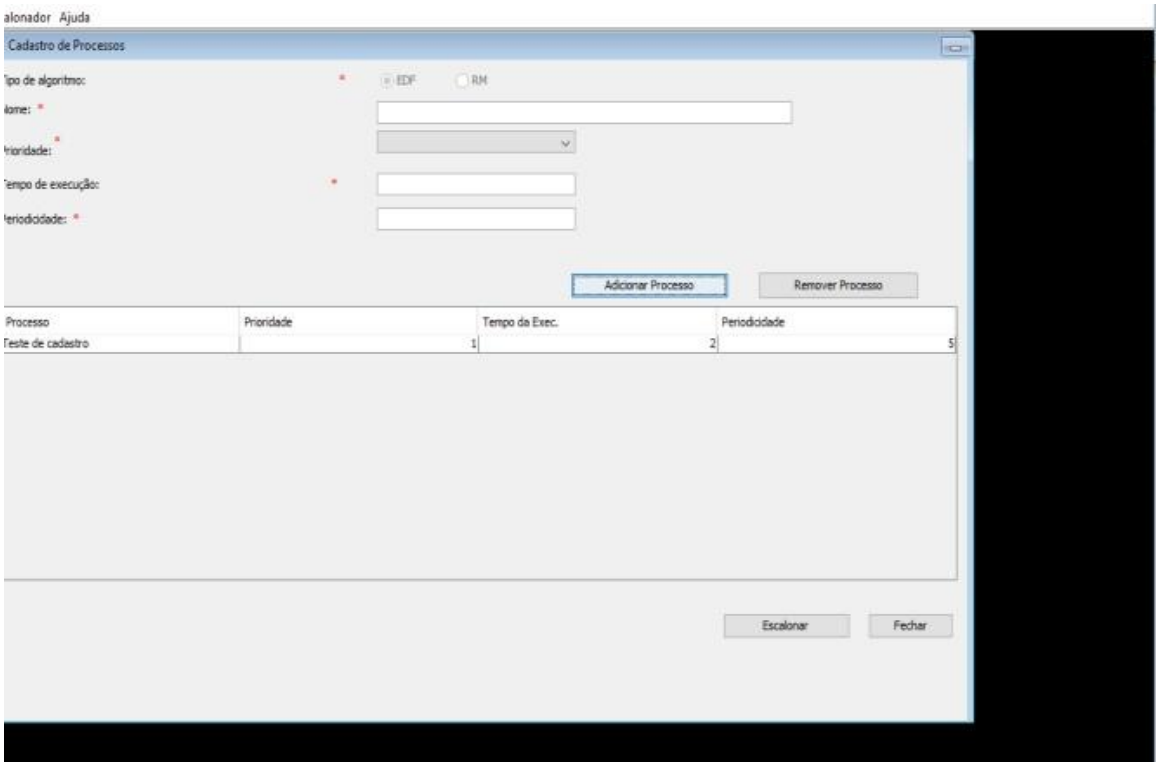

Figura 11: Tela de cadastro de tarefas para o escalonamento RM.

Após o preenchimento das informações, é necessário a adicionar o processo cadastrado na lista, para uma verificação posterior se todos os processos cadastrados serão possíveis de ser escalonado.

O usuário cadastra todas as tarefas que podem ser ou não escalonáveis e ao inicializar o escalonamento, o sistema irá realizar a ordenação da lista, dependendo do algoritmo selecionado. A cada tipo de algoritmo escalonamento informado para a execução de todas as tarefas, o sistema se comportará de forma diferente para a ordenação e execução das tarefas cadastradas.

Após, irá verificar se todas as tarefas cadastradas são passiveis de escalamento. Caso alguma não seja, o mesmo será retirado da fila de processo, garantido assim que todos os processos sejam escalonados no seu período, não havendo nenhuma perda de *deadline*.

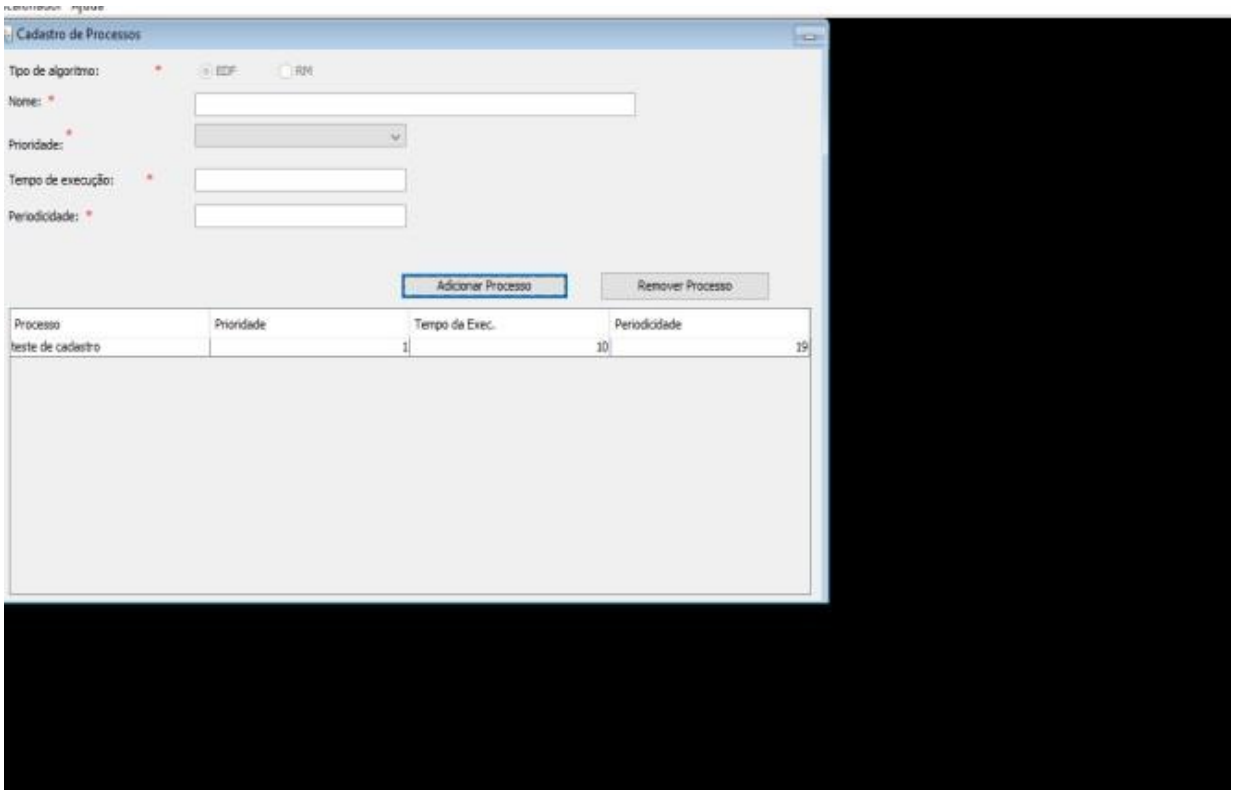

Figura 12: Tela após cadastro de um processo.

A verificação será efetuada através dos valores digitados para cada processo, se o está de acordo com o tipo de algoritmo de escalonamento selecionado. Ao escolher o escalonamento EDF, o escalonador irá calcular, uma a uma, a utilização do processador do conjunto de tarefas cadastradas e o total dele deve ser menor ou igual a 1, caso a *utilização* após a verificação ultrapasse o valor, a tarefa que ocasionou esse desvio não será escalonada e retira do escalonamento. Isso é feito para que seja garantido escalonamento de todas as tarefas (dito um teste de admissão para a execução das tarefas), sem que nenhuma delas perca o *deadline* e sejam executadas corretamente.

No caso do RM, o tempo de utilização do processador deve ser menor ou igual a 0.69 para ter garantido o escalonamento. Caso um processo ao ser adicionado não cumpra essa premissa, o mesmo será retirado da fila de processo que serão escalonados. Isso é feito para garantir que as tarefas que forem escalonáveis, sejam executadas garantindo as premissas do escalonamento de tempo real.

Depois de realizar a ordenação e verificar se elas são passíveis de escalonamento, é dado o inicio da execução do escalonamento das tarefas.

Se o algoritmo executado for EDF e durante o processamento da tarefa, chegar outra tarefa com prioridade maior, o escalonador irá realizar uma preempção das tarefas, parando a que estava em processamento, porém, tem menor prioridade, e iniciando a executar a de maior prioridade.

Ao realizar essa preempção, o processo que estava em execução será retirado da fila de pronto e será adicionada a fila de tarefas suspensas, que ao término do processamento das tarefas de maiores prioridades, essa tarefa é adicionada a fila de pronto e é escalonado partir do ponto em que foi suspensa a execução.

Na execução é apresenta as informações do processo que está sendo executado como nome, tempo de execução, período e vezes executadas. E variado o campo tempo de execução de acordo com o tempo decorrido do processamento.

Ao término da execução de todos os processos que podem ser escalonáveis, o escalonador apresentará as informações dos processos que foram escalonados.

As classes que compõem o escalonador são: Processo, Util, Ativacao, MMC, Escalonador, FilaProcesso, ExecucaoRestanto e Escalonamento.

Na classe Processo contém todas as informações referente ao processo, como o nome, tempo de execução, tempo da próxima ativação e prioridade. A classe Ativacao é definida quando o processo será ativado. Na classe Escalonador é feita a verificação de quais processos cadastrados pelo usuário são escalonáveis ou não, caso o calculo no *hiperperíodo* ultrapasse o valor predeterminado para o tipo de algoritmo selecionado para o escalonamento, o processo não será inserido na lista de processos escalonáveis.

Na classe FilaProcesso, são inseridos os processos numa fila, ordenados de acordo com o algoritmo de escalonamento escolhido e retira do processo da fila, quando o mesmo for executado todas as vezes que foi pré-definida. Além disso, na classe consta o método de exibição das informações que compõem a lista e remoção do processo após a conclusão do processamento.

As classes Util e MMC calculam parâmetros para verificar a admissibilidade de execução dos algoritmos RM e EDF.

O processo de armazenamento do processo que é retirado da execução para dar lugar a execução do processo de maior prioridade é feito na classe ExecucaoRestante.

E por fim temos a classe Escalonamento, a principal do sistema, responsável pelo escalonamento dos processos. Ela verifica se os processos para serem escalonados e quando será a ativação do processo, garantindo assim o escalonamento no tempo correto. Verifica também quando o próximo será ativado e se a prioridade dele é maior do que o processo que está sendo escalonado, para que no momento da ativação do próximo processo, o sistema realize a preempção do processo atual pelo novo processo que acaba de ser ativado.

Além disso, da execução dos processos que estão na fila de pronto, a classe Escalonamento verifica se após o termino da execução de todos os processos na fila de prontos foram executados e se existem processos na fila de processos suspensos, caso sim, ele adiciona para a fila de prontos e executa todo o processo novamente. A Figura 13 apresenta o diagrama de classe do escalonador de tarefas de tempo real desenvolvido.

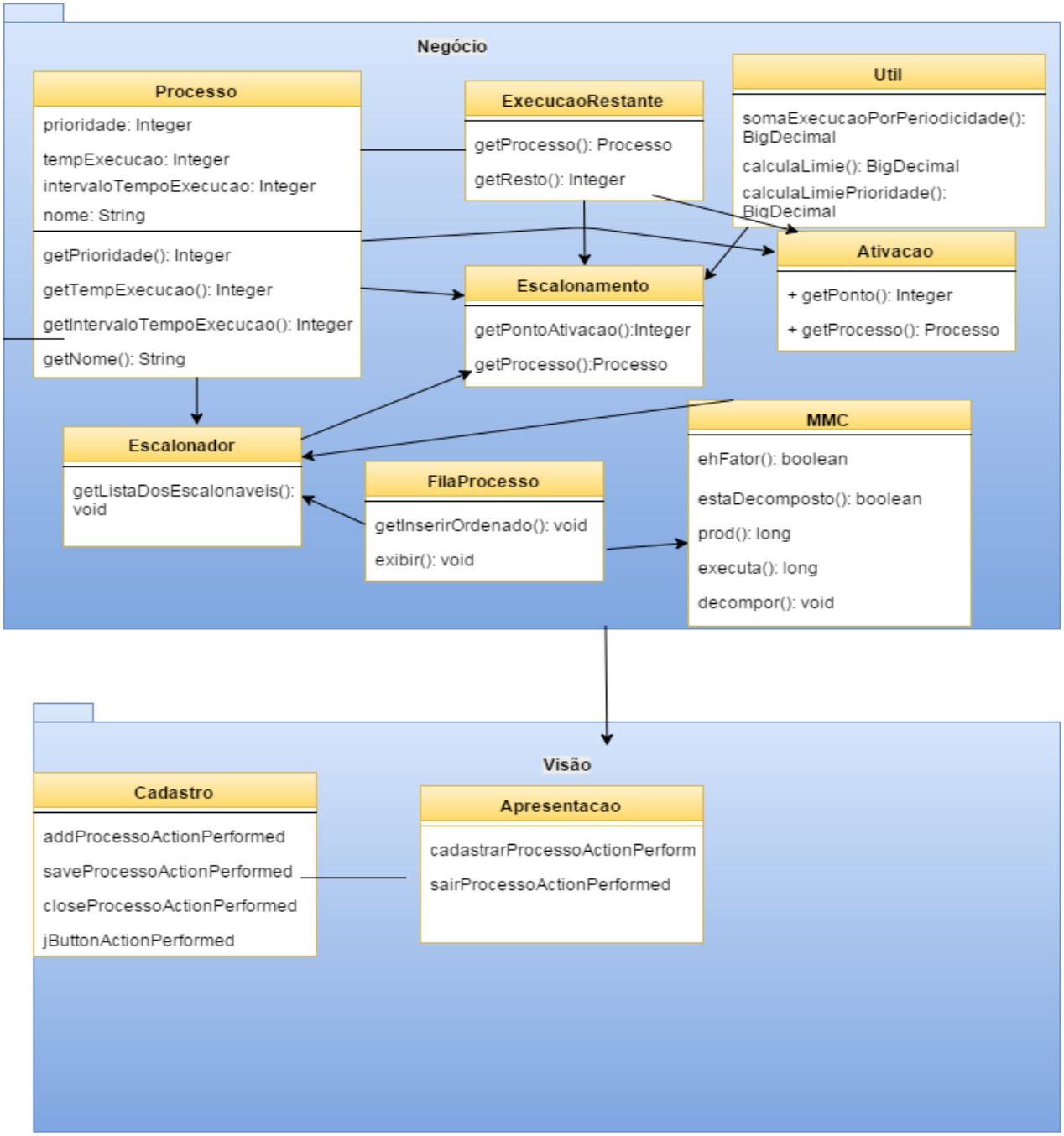

Figura 13: Diagrama de classe

As informações que serão exibidas na tela são as mesmas informações que foram cadastradas para cada processo, como por exemplo, o nome, tempo de início e custo de execução do processo que está sendo executado.

No término do processamento, que ocorre quando o tempo de execução e quando as quantidades de vezes que o processo deve ser executado forem iguais à zero. Quando isso acontecer, o mesmo será retirado da fila dando continuidade ao processo escalonamento de outros processos, caso haja, se não é finalizado o escalonamento.

Para averiguar se o escalonamento está funcionando corretamente, testes de funcionalidade devem ser realizados. Dentre estes testes, enumera-se:

I. Cadastros e execução de tarefas e verificação se a execução foi efetuada corretamente;

II. Cadastros de tarefas que não podem ser escalonados pelos algoritmos de escalonamento EDF e RM;

III. Cadastro de diversas tarefas com intersecção de tempo e verificação da execução da tarefa de menor prioridade. No caso do EDF e com menor tempo de execução no caso do RM.

Para a realização dos testes serão cadastradas as tarefas com as informações exemplos escalonamento do algoritmo RM e EDF e será averiguado se o escalonamento será efetuado corretamente. Logo após será efetuado outros processos de escalonamento para verificar como a ferramenta irá se comportar.

Na execução dos testes do algoritmo RM foram cadastrados 2 processos, Processos 1 e 2, com os dados que são passíveis de escalonamento para verificar se o sistema desenvolvido iria escalonar todas os processos (ver Tabela 1).

Processo	Tempo de execução	Periodicidade
Processo 1	1	3
Processo 2	2	5

Tabela 1: Dados dos processos cadastrados para RM

Após o cadastro e a inicialização do escalonamento foi apresentado no log qual processo estava sendo executada, a fatia de tempo de sua execução e quando a execução do processo foi inicializada, como mostra a Figura 14.

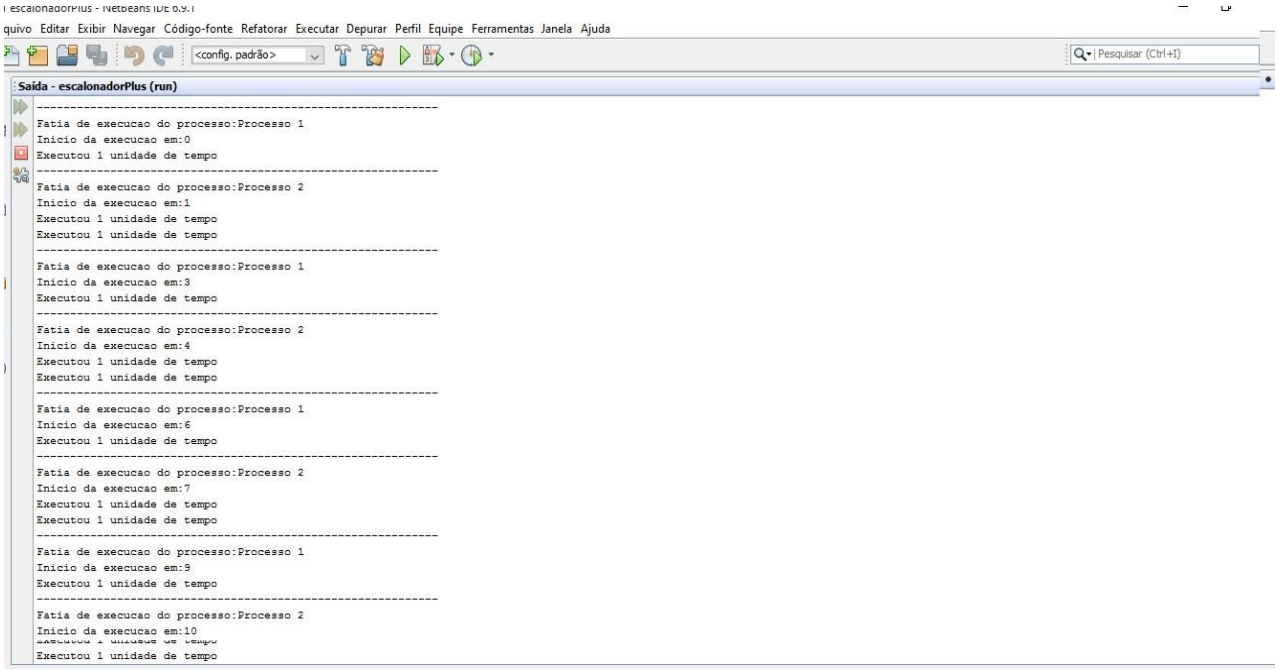

Figura 14: Log de execução de processo RM cadastrados

Nos testes do algoritmo EDF foram cadastrados 3 processos, Processos A, B e C, onde C tem prioridade maior que os outros processos e deve ser executado primeiro e após o termino de processamento do C, o sistema deve executar o processo B, pois ele é o segundo processo com maior prioridade e consequentemente o último processo que deve ser executado é o processo A (ver Tabela 2).

Processo	Prioridade	Tempo de execução	Periodicidade
Processo A	3	4	13
Processo B	2	3	5
Processo C	1	5	9

Tabela 2: Dados dos processos cadastrados para EDF

Na execução o sistema desenvolvido como já era de se esperar, executou primeiro o processo C, respeitando a premissa do algoritmo EDF, executando primeiro quem tem a maior prioridade e após a execução dele, foi executado o processo B e logo após o A, como mostra a Figura 15.

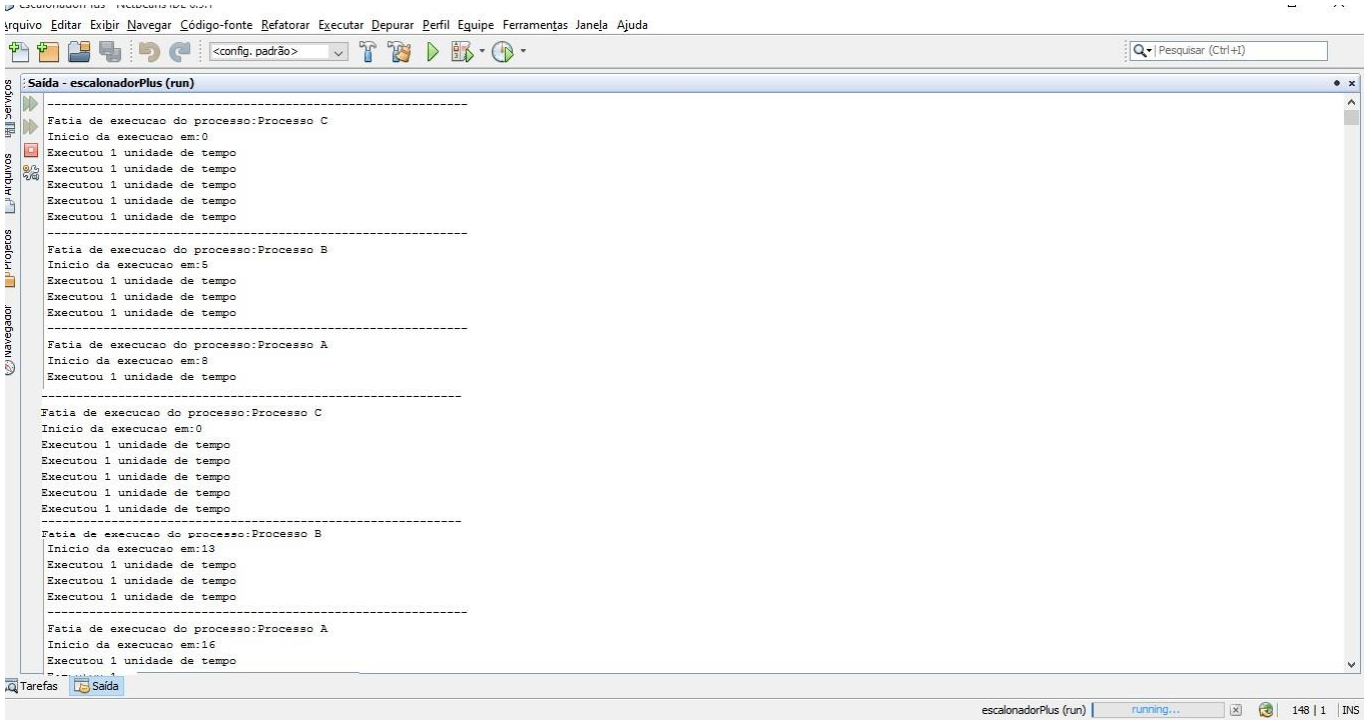

Figura 15: Log de execução de processos EDF cadastrados

Além desses testes, foi realizado também o teste de preempção e onde além de ser considerada a prioridade das tarefas, é levada em consideração também a verificação de qual processo está para perder o *deadline*, e caso houve um processo como essas coisas, mesmo tendo outro processo cadastrado com prioridade maior, o escalonador.

Para isso foram cadastradas duas tarefas, que quando o processo B irá perder o *deadline*, o escalonador desenvolvido continua a execução para que o processo não perca o *deadline* (ver Tabela 3).

Processo	Prioridade	Tempo de execução	Periodicidade
A	10	20	20
B	25	50	50

Tabela 3: Dados dos processos cadastrados para EDF

Ao iniciar o processo de escalonamento, o escalonador executa primeiro a tarefa C, utiliza o processador do tempo 0 até 5, logo após, o sistema executa parte do processo B, pois chegou o processo A para ser escalonado novamente, o escalonador, causa preempção, e executa o processo A, como mostra a Figura 15.

Após o término da execução do processo A, o escalonador volta a dar continuidade à execução do processo B. Apesar de que o tempo de inicialização do processo A chegou novamente, o escalonador dessa vez não causa a preempção, pois se isso fosse realizado o processo B, perderia o *deadline*, então, o escalonador continua a execução da tarefa B até a sua conclusão e depois inicializa a execução do processo A, como é apresentada na Figura 16.

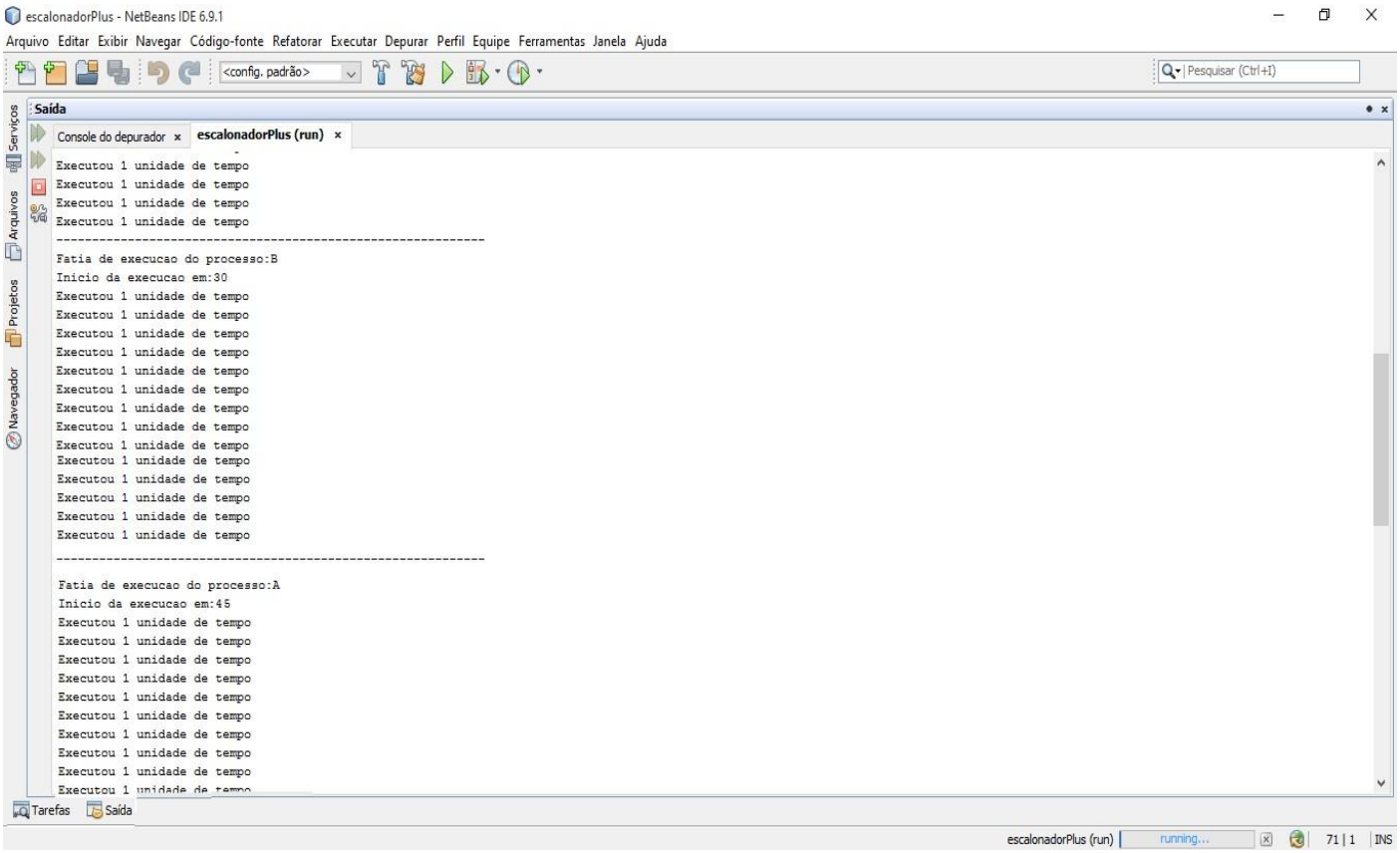

Figura 16: Log de execução de processos EDF cadastrados

8. CONCLUSÃO E TRABALHOS FUTUROS

Neste trabalho foi apresentada uma ferramenta de escalonamento de tempo real para integração com simuladores de sistemas distribuídos, ajudando assim a melhorar a precisão e dando mais uma função ao simulador, que além de realizar simulação de sistemas distribuídos, será possível realizar com tempo real.

A simulação de sistemas é importante, pois, através dela é definida a construção de um programa computacional para implementar modelos de fenômenos ou sistemas dinâmicos. Além disso, nos sistemas distribuídos possibilita o desenvolvimento de aplicações tolerantes a falhas, por meio de mecanismos como o da replicação de processos, garantido a continuidade do serviço em execução mesmo que ocorrem defeitos.

A ferramenta foi idealizada para complementar as funções dos simuladores, ajudando assim na implementação de modelos e na simulação de sistemas distribuídos de tempo real, efetuando o escalonamento dos processos cadastrados para simulação.

A princípio a ferramenta de escalonamento foi desenvolvida para executar dois tipos de algoritmos de escalonamento, os mais utilizados, que são os algoritmos RM e EDF, permitindo ao usuário cadastrar os processos informando se tem ou não prioridade, e de acordo com os parâmetros informados pelo usuário e as premissas de cada algoritmo, será efetuado o processo de escalonamento.

Após o processo de cadastramento e inicialização da execução, o sistema executa somente os processos que são passíveis de escalonamento e caso algum deles não cumpra as premissas do algoritmo selecionado, será retirado da fila de processamento e não será executado.

Os processos que estão sendo escalonados são apresentados em log, com as suas informações, deixando claro para o usuário qual o processo está sendo executada, quanta fatia de tempo foi consumida.

A ferramenta proposta deverá ser integrada posteriormente com o simulador HDDSS, de modo a habilitar o uso do escalonamento periódico de tempo real neste simulador de sistemas distribuídos.

Como trabalhos futuro, tem-se:

I. Vincular o escalonador a um simulador de sistemas distribuídos;

II. Para melhor visualização das informações dos processos escalonados, implementar um relatório gráfico para as tarefas que foram e estão sendo executadas;

III. Implementar um escalonamento multi processado, que execute vários algoritmos de escalonamento na mesma fila de processos;

IV. Escalonar processos periódicos.

9. REFERÊNCIAS

[1] FREITAS, Allan Edgard Silva; MACÊDO, Raimundo José de Araújo. A performance evaluation tool for hybrid and dynamic distributed systems. ACM SIGOPS Operating Systems Review, v. 48, n. 1, p. 11-18, 2014.

[2] MACÊDO, Raimundo et al. Tratando a Previsibilidade em Sistemas de Tempo-real Distribuídos: Especificação. Linguagens, Middleware e Mecanismos Básicos (minicurso). XXII Simpósio Brasileiro de Redes de Computadores. Gramado, RS, Brasil, 2004.

[3] LYNCH, Nancy A. Distributed algorithms. Morgan Kaufmann,
1996.

[4] FREITAS, ALLAN EDGARD SILVA. SIMULAÇÃO DE SISTEMAS DISTRIBUÍDOS HÍBRIDOS E DINÂMICOS. 2013. Tese de Doutorado. Universidade Federal da Bahia.

[5] AVIŽIENIS, Algirdas et al. Basic concepts and taxonomy of dependable and secure computing. Dependable and Secure Computing, IEEE Transactions on, v. 1, n. 1, p. 11-33, 2004.

[6] CRISTIAN, Flavin. Understanding fault-tolerant distributed systems. Communications of the ACM, v. 34, n. 2, p. 56-78, 1991.

[7] TRINDADE, Renata de Moraes. Uso do network simulator- NS para simulação de sistemas distribuídos em cenários com defeitos. 2003. Tese de Doutorado. UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL.

[8] FARINES, Jean-Marie; FRAGA, Joni da Silva; OLIVEIRA, RS de. Sistemas de Tempo Real. Escola de Computação, v. 2000, p. 201, 2000.

[9] MEIRA, Marcos. Política de Escalonamento de processos em Linux. Campo Digital, v. 3, n. 1, 2011.

[10] PEDRO, Antonio; SANTOS, Max; RENÓ, Demétrio. Análise de escalonabilidade de tarefas no kernel de tempo real S. Ha. RK. In: Anais do III Congresso Brasileiro de Computação (CBCOMP), Universidade do Vale do Itajaí, Itajaí, SC.

[11] OLIVEIRA, Arnaldo SR; ALMEIDA, Luís. Ensaio sobre os sistemas de tempo real. Electrónica e Telecomunicações, v. 4, n. 2, p. 261-266, 2004.

[12] CECHIN, Sergio; NETTO, João. Ferramentas de avaliaçao de desempenho de sistemas computacionais. ESCOLA REGIONAL DE ALTO DESEMPENHO, ERAD, 1ª, p. 127-150, 2001.

[13] SANTANA, R. H. C. et al. Técnicas para avaliação de desempenho de sistemas computacionais. Notas didáticas do ICMC, ICMC–USP, 1994.

[14] MACIEL, Paulo RM; LINS, Rafael D.; CUNHA, Paulo RF. Introdução às redes de Petri e aplicações. UNICAMP-Instituto de Computacao, 1996.

[15] RODRIGUES, Cássio Leonardo. Verificação de modelos em redes de petri orientadas a objetos. 2004. Tese de Doutorado. Universidade Federal de Campina Grande.

[16] MOURA, Raimundo Santos. Metodologia para modelagem, validação e programação de controladores lógicos industriais usando statecharts básicos. 2009. Tese de Doutorado. Universidade Federal do Rio Grande do Norte.

[17] ALVAREZ, Guillermo A.; CRISTIAN, Flaviu. Simulation-based testing of communication protocols for dependable embedded systems. The Journal of Supercomputing, v. 16, n. 1-2, p. 93-116, 2000.

[18] MACDOUGALL, Myron H. Simulating computer systems: techniques and tools. MIT press, 1989.

[19] BRESLAU, Lee et al. Advances in network simulation. Computer, n. 5, p. 59-67, 2000.

[20] BARCELLOS, Marinho P. et al. Beyond network simulators: Fostering novel distributed applications and protocols through extendible design. Journal of Network and Computer Applications, v. 35, n. 1, p. 328-339, 2012.

[21] VARGA, András et al. The OMNeT++ discrete event simulation system. In: Proceedings of the European simulation multiconference (ESM'2001). sn, 2001. p. 65.

[22] URBAN, Peter; DÉFAGO, Xavier; SCHIPER, André. Neko: A single environment to simulate and prototype distributed algorithms. In: Information Networking, 2001. Proceedings. 15th International Conference on. IEEE, 2001. p. 503-511.

[23] AUDSLEY, Neil C.. et al. STRESS: A simulator for hard real- time systems. Software: Practice and Experience, v. 24, n. 6, p. 543-564, 1994.

[24] CASILE, Antonino et al. Simulation and tracing of hybrid task sets on distributed systems. In: Real-Time Computing Systems and Applications, 1998. Proceedings. Fifth International Conference on. IEEE, 1998. p. 249-256.

[25] CRUZ, Gisélia Magalhaes; LIMA, George. Simulador de escalonamento para sistemas de tempo real. IV WTICG (Trabalho de Conclusao de Curso)-ERBASE, p. 1-11, 2006.