ANÁLISE DE ESCALONABILIDADE DE TAREFAS NO KERNEL DE TEMPO REAL S.HA.R.K.

Antonio PEDRO antoniopedro@unilestemg.br
Max SANTOS maxmauro@unilestemg.br
Demétrio RENÓ reno@unilestemg.br

Antonio PEDRO
Centro Universitário do Leste de Minas Gerais – UnilesteMG
Av. Tancredo Neves nº 3500 – Bairro Universitário
Coronel Fabriciano – Minas Gerais – Brasil – CEP 35170-056
Tel.: 55 31 3842-6325
http://professores.unilestemg.br/ltr/antonio.html

Max SANTOS
Centro Universitário do Leste de Minas Gerais – UnilesteMG
Av. Tancredo Neves nº 3500 – Bairro Universitário
Coronel Fabriciano – Minas Gerais – Brasil – CEP 35170-056
Tel.: 55 31 3842-6325
http://professores.unilestemg.br/ltr/max.html

Demétrio RENÓ
Centro Universitário do Leste de Minas Gerais – UnilesteMG
Av. Tancredo Neves nº 3500 – Bairro Universitário
Coronel Fabriciano – Minas Gerais – Brasil – CEP 35170-056
Tel.: 55 31 3842-6325

RESUMO

Este trabalho apresenta um teste da escalonabilidade de tarefas em tempo real no Kernel do S.Ha.R.K usando os algoritmos Rate Monotonic e Earliest Deadline First. Serão apresentados dois experimentos, independentes onde, no primeiro, é considerado um conjunto com três tarefas periódicas e com deadline igual ao período. Neste conjunto, a primeira tarefa faz um cálculo exponencial de um valor float, a segunda tarefa realiza um cálculo para determinar a posição em que um texto será exibido e em seguida o imprime na tela do monitor e na terceira tarefa é exibido um texto simples em uma posição estática, sem nenhum cálculo. Já para o segundo experimento, será considerado um conjunto com cinco tarefas periódicas, com deadline também igual ao período. Todas as cinco tarefas realizam o mesmo procedimento: uma bola se movimenta pelo monitor, porém dentro de um espaço pré-definido. Os dados das tarefas poderão ser visualizados através de gráficos da aplicação. Espera-se comprovar o funcionamento do escalonador do S.Ha.R.K., através de conjuntos de tarefas que sejam escalonáveis pelo algoritmo Earliest Deadline First e não escalonáveis pelo Rate Monotonic.

Palavras-chave: Tempo Real, S.Ha.R.K., Escalonador de tarefas, Deadline.

ABSTRACT

This paper presents a test of scheduling of tasks in S.Ha.R.K Real Time Kernel, using the algorithms Rate Monotonic and Earliest Deadline First. Two independent experiments will be presented, where in the first task set, three periodic tasks are considered, and with deadline equal to the period. In this set, the first task makes an exponential calculation of a value float, the second task carries through a calculation to define the position of text will be show and after impress the text in the monitor and in the third task is shown a simple text in a static position, without no type of calculation. For the secound experiment, a set with five periodic tasks will be considered, with deadline also equal to the period. All the five tasks carry through procedure the same: a ball moves for the monitor, however inside of a daily pay-define space. The data of the tasks could be visualized through graphs of the application. One expects to prove the functioning of the S.Ha.R.K. schedulling, through task set that is scheduled for the algorithm Earliest Deadline First and not schedule for the Rate Monotonic.

Key-words: Real-Time, S.Ha.R.K., Scheduler of Tasks, Deadline.

ANÁLISE DE ESCALONABILIDADE DE TAREFAS NO KERNEL DE TEMPO REAL S.HA.R.K.

RESUMO

Este trabalho apresenta um teste da escalonabilidade de tarefas em tempo real no Kernel do S.Ha.R.K usando os algoritmos Rate Monotonic e Earliest Deadline First. Serão apresentados dois experimentos independentes onde, no primeiro, é considerado um conjunto com três tarefas periódicas e com deadline igual ao período. Neste conjunto, a primeira tarefa faz um cálculo exponencial de um valor float, a segunda tarefa realiza um cálculo matemático para determinar a posição em que um texto será exibido e em seguida o imprime na tela do monitor e na terceira tarefa é exibido um texto simples em uma posição estática, sem nenhum cálculo. Já para o segundo experimento, será considerado um conjunto com cinco tarefas periódicas, com deadline também igual ao período. Todas as cinco tarefas realizam o mesmo procedimento: uma bola se movimenta pelo monitor, porém dentro de um espaço pré-definido. Os dados das tarefas poderão ser visualizados através de gráficos da aplicação. Espera-se comprovar o funcionamento do escalonador do S.Ha.R.K., através de conjuntos de tarefas que sejam escalonáveis pelo algoritmo Earliest Deadline First e não escalonáveis pelo Rate Monotonic.

Palavras-chave: Tempo Real, S.Ha.R.K., Escalonador de tarefas, Deadline.

ABSTRACT

This paper presents a test of scheduling of tasks in S.Ha.R.K Real Time Kernel, using the algorithms Rate Monotonic and Earliest Deadline First. Two independent experiments will be presented, where in the first task set, three periodic tasks are considered, with deadline equal to the period. In this set, the first task makes an exponential calculation of a value float, the second task carries through a calculation to define the position of text will be show and after impress the text in the monitor and in the third task is shown a simple text in a static position, without no type of calculation. For the secound experiment, a set with five periodic tasks will be considered, with deadline also equal to the period. All the five tasks carry through procedure the same: a ball moves for the monitor, however inside of a daily pay-define space. The data of the tasks could be visualized through graphs of the application. One expects to prove the functioning of the S.Ha.R.K. schedulling, through task set that is scheduled for the algorithm Earliest Deadline First and not schedule for the Rate Monotonic.

Key-words: Real-Time, S.Ha.R.K., Scheduler of Tasks, Deadline.

1 Introdução

Sistema em Tempo Real (STR) é um sistema computacional que deve reagir a estímulos oriundos de seu ambiente em prazos específicos. Estão, portanto sujeitos a requisitos associados à passagem de tempo. O atendimento a tais requisitos faz com que o sistema deva fornecer seus resultados corretos em um prazo específico. Compreendem assim uma classe de sistemas em que além de ter de realizar sua execução de forma logicamente correta, deverá ainda ter sua execução

finalizada antes de um prazo definido. Caso o resultado do processamento seja fornecido após o limite especificado, poderá não ser mais necessário, ou até mesmo representar uma ameaça ao funcionamento do sistema como um todo. As áreas de atuação dos STR vão desde pequenos controladores de eletrodomésticos até sistemas de controle de tráfego aéreo e/ou ferroviário. Outra utilização em que os STR estão cada vez mais imersos é a área de aplicações multimídia, sendo utilizado principalmente em vídeo-conferência. Para que se torne possível utilizar um STR para atividades de controle com requisitos temporais, é necessário que se utilize metodologias adequadas para o seu correto funcionamento. Atualmente, são utilizadas algumas ferramentas convencionais para o desenvolvimento de STR, o que não prevê um tratamento explícito das variáveis de tempo. Podemos, no entanto considerar como suficiente a utilização de um STR que suporte interrupções, concorrência e mecanismos de temporização com granularidade adequada (BURNS; WELLINGS, 1997). A utilização de metodologias e ferramentas convencionais, tem sido consideradas suficientes para o desenvolvimento de STR de baixa complexidade, para os quais não se colocam exigências temporais rigorosas. No entanto, as aplicações desenvolvidas poderão ter um comportamento temporal imprevisível, não oferecendo uma adequada garantia do seu correto funcionamento. Para projetos com maior nível de exigência, é fundamental a utilização de metodologias de desenvolvimento que considerem de uma forma adequada os requisitos temporais ao longo de todo o processo de desenvolvimento (JONES, 1997).

Este artigo está organizado em 5 itens: No primeiro, é realizada uma introdução aos STR. O item 2 traz os conceitos básicos dos sistemas de tempo real, o item 3 mostra uma visão geral do Kernel do S.Ha.R.K, o item 4 descreve os detalhes do experimento juntamente com os resultados obtidos e finalmente o item 5 traz as conclusões e perspectivas futuras de trabalho.

2 – Conceitos Básicos de Tempo Real

Em um STR, uma tarefa consiste em um segmento de código cuja execução possui uma restrição temporal. As restrições temporais estão intimamente ligadas ao deadline das tarefas do sistema. Entende-se por deadline o prazo limite para que uma tarefa seja totalmente computada. O deadline é iniciado quando a tarefa for ativada, mesmo que o início de sua computação não coincida com o de sua ativação. A figura 1 permite um melhor entendimento do ciclo de vida de uma tarefa.

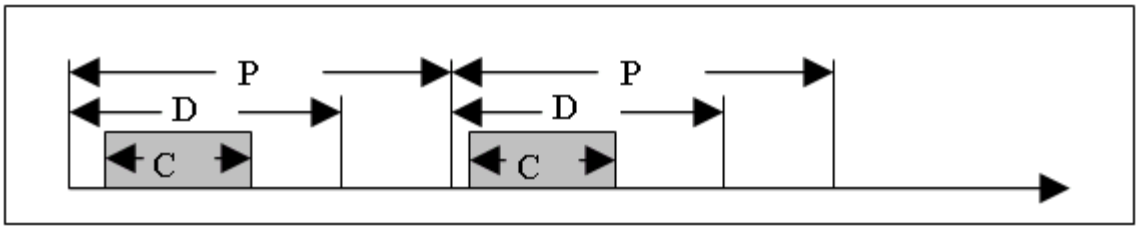

Figura 01 – Ciclo de vida de uma tarefa

Quanto a forma de ativação, as tarefas podem ser periódicas, esporádicas ou aperiódica. As tarefas periódicas possuem um tempo de ativação pré-determinado. Nas tarefas esporádicas seu tempo de ativação não é determinado, mas se conhece um intervalo mínimo entre suas ativações. As tarefas aperiódicas são ativadas por algum estímulo externo ao sistema.

Os STR podem ser classificados em sistemas críticos onde conseqüências graves poderão ocorrer ao meio se o deadline for rompido, e não críticos, onde a performance do sistema é prejudicada, mas não causará conseqüências graves ao meio (FARINES; FRAGA; OLIVEIRA, 2000).

2.1 O escalonador de Tarefas

O escalonamento de tarefas, em um STR, é implementado pelo escalonador de tarefas. Entende-se por escalonar uma tarefa, o ato de ordenar as tarefas que estão prontas para serem executadas e enviá-las ao processador (RAMAMRITHAN; STANKOVIC, 1994). A forma como o escalonador ordena as tarefas, irá de acordo com o algoritmo de escalonamento que está sendo usado. Para tornar possível esta ordenação, o escalonador atribui prioridades às tarefas de um STR, prioridades estas que permitirão que tarefas críticas possam ser computadas antes das tarefas menos críticas.

Digamos que a tarefa que tenha prioridade de índice menor seja a mais prioritária do conjunto de tarefas. De acordo com o algoritmo usado pelo escalonador, as tarefas são classificadas em ordem de prioridades. Esta classificação poderá ser estática ou dinâmica. Em um algoritmo de escalonamento de prioridade estática, todas as tarefas são classificadas ainda em fase de projeto, levando-se em conta fatores como o deadline, o período de ativação, entre outros. Já para algoritmos de prioridades dinâmicas, as tarefas são classificadas em tempo de execução, levando-se em consideração diversos outros fatores de acordo com o algoritmo.

2.2 Algoritmos de Tempo Real mais Comuns

Neste tópico será apresentado o funcionamento dos algoritmos de escalonamento mais comuns.

Algoritmos de escalonamento são as estratégias usadas para ordenação das tarefas antes de enviá-las ao processador. Um bom escalonador garante que para todo conjunto escalonável de tarefas sempre encontre um escalonamento.

Os algoritmos estáticos permitem garantir que requisitos temporais sejam garantidos, sendo portando bons para STR críticos, porém nem sempre é possível implementá-los, pois na prática os algoritmos estáticos não suportam escalonar muitos conjuntos de tarefas que são escalonados por algoritmos dinâmicos. Em seguida serão apresentados os algoritmos de escalonamento mais comuns (PINEDO, 1995).

2.2.1 - Rate Monotonic (Taxa Monotônica) - RM

Este algoritmo baseia-se na periodicidade (intervalo em que uma tarefa solicita ser escalonada) das tarefas, sendo que quanto menor a periodicidade, maior sua prioridade no conjunto de tarefas. É um ótimo algoritmo estático e que exige um escalonador preemptivo.

Premissas simples para facilitar a análise de escalonabilidade do RM definem que:

· As tarefas são sempre periódicas e independentes;

· O deadline de cada tarefa é igual ao seu período (Di = Pi);

· O tempo de computação (tempo gasto para computar uma tarefa) de cada tarefa é constante e conhecido.

Porém, nem sempre é possível atender a estas premissas na prática, pois muitas aplicações de STR possuem tarefas aperiódicas e dependentes, mas é suficiente para um entendimento do RM.

Para se garantir que um conjunto de tarefas seja escalonável, é necessário efetuar um cálculo apresentado em (LIU; LAYLAND, 1973). A equação (1) demonstra como é realizado este cálculo, considerando-se um deadline igual ao período

$$U = \sum_{i=1}^{n} \frac{C_i}{P_i} \leq n\left(\sqrt[n]{2} - 1\right) \quad (1)$$

onde: U é a taxa de utilização do processador, C Tempo de computação da tarefa $_{i,}$ P é a periodicidade da tarefa $_{i}$ e n é o número de tarefas existentes no conjunto.

Para que o conjunto seja escalonável, a taxa de utilização do processador (lado esquerdo da equação) deverá ser menor que o valor resultado pelo lado direito da equação.

2.2.2 - Deadline Monotonic (Deadline Monotônica) - DM

Parecido com o RM, o DM se baseia no deadline da tarefa, priorizando em ordem reversa, ou

seja, quanto menor o deadline, maior a prioridade da tarefa. O cálculo para garantir que uma determinada tarefa seja escalonável considerando-se um deadline menor que o período, está demonstrado abaixo pela equação (2):

$$R_i^{n+1} = C_i + \sum \left\lceil \frac{R_i}{P_i} \right\rceil C_i \qquad (2)$$

onde: R é o tempo máximo de resposta da tarefa $_i$.

A solução da equação (2) é definida de forma interativa com o valor inicial de R_i sendo zero e a solução ocorrerá quando a equação recursiva convergir, com uma utilização do processador menor que 100%. Para que uma tarefa seja escalonável, seu tempo de resposta máximo deverá ser menor que seu deadline.

2.2.3 - Earliest Deadline First (Deadline mais Adiantado Primeiro) - EDF

O EDF é um algoritmo dinâmico que em tempo de execução da aplicação, ordena as tarefas baseando-se em um cálculo, onde a cada instante verifica-se qual tarefa está com o seu deadline mais próximo de ser ultrapassado. Baseado neste cálculo, esta tarefa terá maior prioridade.

O cálculo suficiente para garantir a escalonabilidade de um conjunto de tarefas pelo algoritmo EDF, considerando-se um deadline igual ao período está disposto na equação (3):

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1 \qquad (3)$$

Para que o conjunto de tarefas seja escalonável, basta que a taxa de utilização do processador seja menor do que 1.

No próximo item, será detalhado o funcionamento do Kernel do S.Ha.R.K., Kernel este que pode implementar todos os algoritmos de escalonamento citados neste tópico.

3 O Kernel de Tempo Real S.Ha.R.K.

O sistema S.Ha.R.K. desenvolvido pela universidade de Pisa, Itália, é uma arquitetura de Kernel configurada dinamicamente, designada para suportar aplicações de Tempo Real críticas, não criticas e de não Tempo Real, com algoritmos de escalonamento configurados em tempo de execução.

O Kernel é inteiramente modular nos termos de políticas de escalonamento, servidores aperiódicos, e de controle de protocolos correntes, que geralmente são não modulares em Sistemas Operacionais tradicionais. A modularidade é conseguida dividindo as atividades do

sistema entre um Kernel genérico e um módulo conjunto, que podem ser registrados no tempo de iniciação para configurar o Kernel de acordo com os requerimentos de uma aplicação específica.
Os maiores beneficiados para a arquitetura de Kernel proposta são as aplicações, que poderão ser desenvolvidas independentemente de uma configuração de sistema específico. Os novos módulos podem ser adicionados ou substituídos de uma mesma aplicação, para avaliar os efeitos de políticas de escalonamento específicas em termos de prioridade, overhead, e desempenho (GAI, 2001).

3.1 Iniciação da aplicação

Uma aplicação S.Ha.R.K., ao ser iniciada, antes de executar o modo de multi-tarefamento (várias tarefas concorrendo ao processador), um arquivo de iniciação é chamado. Neste arquivo, o Kernel faz uma chamada a função *__kernel_register_levels__()*. Esta função é responsável por carregar os módulos necessários à aplicação, como módulos de escalonamento (RM, EDF, etc.), módulos de recursos como protocolos de acesso a recursos compartilhados (Semáforos, Mutexes, etc.), além de outros dispositivos que poderão ser necessários quando o modo de multi-tarefamento for iniciado.

3.2 A arquitetura do S.Ha.R.K.

De um ponto de vista Geral, o S.Ha.R.K. é um Kernel genérico que tem a finalidade de obter independência entre a aplicação e os algoritmos de escalonamento. O Kernel do S.Ha.R.K. não implementa qualquer algoritmo de escalonamento, mas proporciona decisões de escalonamento para entidades externas, os chamados módulos de escalonamento (GAI, 2001, p.128).
O Kernel Genérico fornece mecanismos que permitem que a aplicação se abstraia de algoritmos específicos. Permite também que novos algoritmos sejam implementados e usados em uma aplicação. Esta característica é fornecida pelo Model Mapper que será posteriormente detalhado.
A figura 02 demonstra de forma geral o funcionamento da arquitetura do Kernel:

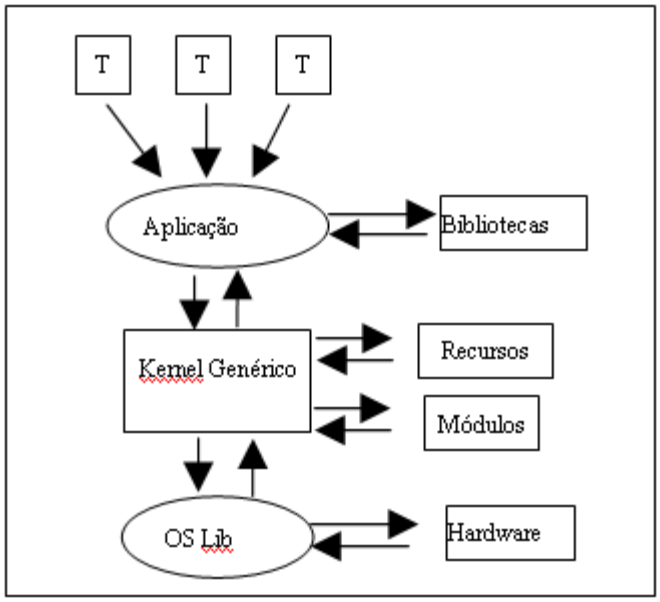

Figura 02 - Arquitetura do Kernel do S.Ha.R.K. – T são as tarefas de uma aplicação.

A comunicação entre o Kernel genérico e o Hardware é fornecida pelo OS LIB (ABENI; LAMASTRA, 1998). Como pode ser visto na figura 02, a aplicação é completamente independente dos módulos de escalonamento. Cada módulo possui uma interface para que seja possível a comunicação com o Kernel genérico, seja no sentido Kernel Genérico - Módulo (solicitar o escalonamento de uma tarefa), seja no sentido Módulo - Kernel Genérico (solicitar ao hardware o temporizador).

Módulos de escalonamento são usados pelo Kernel genérico para escalonar tarefas, ou servir pedidos aperiódicos usando um servidor aperiódico. Os módulos de escalonamento são organizados em níveis, um módulo para cada nível. Estes níveis podem ser vistos como níveis de prioridade de escalonamento (as prioridades são definidas na função de iniciação). Quando o Kernel genérico tem que tomar uma decisão de escalonamento, ele solicita ao módulo para escalonar a tarefa, de acordo com as prioridades fixas: primeiro, ele invoca uma decisão de escalonamento ao módulo de prioridade mais alta, e então (se o módulo não puder escalonar qualquer tarefa pronta e rodar), ele solicita ao próximo módulo de escalonamento de prioridade mais alta, e assim por diante. O Kernel genérico escalona a primeira tarefa de prioridade mais alta sem esvaziar a fila de módulos.

Na prática quando uma tarefa é criada, o Kernel envia ao componente chamado Model Mapper o modelo da tarefa (por exemplo, tarefa crítica ou não crítica) e as suas requisições (QoS). Será o Model Mapper que definirá (de acordo com a política interna da aplicação), a qual módulo será enviada a tarefa para escalonamento, como mostrado na figura 03 abaixo:

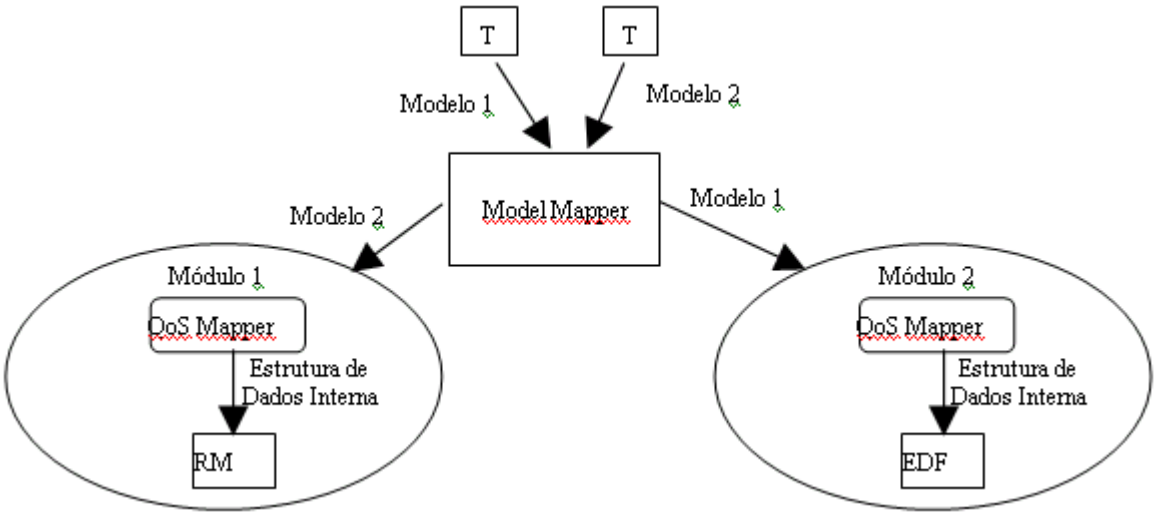

Figura 03 – Funcionamento do Model Mapper

Desta maneira, a política de escalonamento pode ser ajustada simplesmente modificando a função de iniciação.

3.3 Os Módulos de Escalonamento

Os módulos de Escalonamento são organizados em níveis, sendo um módulo para cada nível. O Model Mapper seleciona o módulo para escalonar de acordo com a estratégia de política de prioridades. Sendo assim, uma tarefa que será escalonada por um módulo M2, que tem menor prioridade do que o módulo M1, diz-se que esta tarefa será escalonada em segundo plano em relação a uma outra tarefa escalonada por M1. Como é exemplificado na Figura 04.

Nível 0	Módulo 0
Nível 1	Módulo 1
Nível 2	Módulo 2

Figura 04 – Exemplo da organização dos módulos de escalonamento.

Neste exemplo uma tarefa que é tomada para escalonamento pelo módulo 1, será escalonada em segundo plano com relação às tarefas que são tomadas para escalonamento pelo módulo 0, e em primeiro plano com relação às tarefas que são tomadas para escalonamento pelo módulo 2.

Para poder escalonar as tarefas que lhe são solicitadas, cada módulo possui uma interface. Esta interface permite que diferentes modelos de tarefas possam ser escalonados em um mesmo módulo de escalonamento. Por exemplo, o módulo EDF quando lhe é enviado um modelo de tarefa crítica, que possui os parâmetros período e pior tempo de execução; transforma estes

parâmetros em tempo de reativação e deadlines, que são as variáveis com as quais trabalha.
As funções de interface fornecida pelo módulo de escalonamento podem ser agrupadas em duas classes: funções públicas e privadas. Funções públicas são aquelas que são chamadas diretamente pelo Kernel Genérico. Algumas funções estão diretamente relacionadas ao ciclo de vida da tarefa (como criação, ativação, preemptação, terminação, etc.) outras estão relacionadas ao módulo como um todo (algoritmo de escalonamento, teste de escalonamento, etc.). Funções privadas são as funções em que o módulo disponibiliza para outros módulos a possibilidade de escalonar suas tarefas. As funções privadas são somente chamadas por outras funções públicas ou privadas, e nunca pelo Kernel Genérico (GAI, 2001).

4 Detalhes do Experimento

Neste item, iremos fazer uma análise comparativa entre os testes de escalonamento apresentados no item 2. O esquema usado para o desenvolvimento da aplicação de estudo do escalonamento consiste de dois experimentos. O primeiro experimento possui um conjunto com três tarefas periódicas, independentes, com deadline igual ao período, e escalonadas pelos algoritmos RM e EDF.
Cada uma das tarefas possui suas próprias finalidades, sendo as seguintes:
A tarefa A faz um cálculo exponencial onde tanto a base como o expoente, são valores float, e em seguida atribui-se o resultado a uma variável, com pode ser visto pelo código:

```
a += pow(a,b);
```

A tarefa B exibe um caractere no monitor, sendo que antes disso, é feito um cálculo para definir as posições xy para impressão. O código da tarefa está demonstrado abaixo:

```
if (x <= 10 ) direcao = 1;          /*Limite esquerdo*/
if (x >= 100) direcao = 0;          /*Limite direito*/
if (direcao == 0) x--;              /*movimento para a esquerda*/
   else x++;                        /*movimento para a direita*/
grx_text(“A”, x, 100, 15, 0);       /*impressão*/
```

a função *grx_text(texto, x, y, f, b)*, recebe 5 parâmetros: texto a ser impresso, posição x, posição y, cor de primeiro plano e cor de segundo plano respectivamente.
A tarefa C exibe um texto de nove caracteres em uma posição fixa do monitor:

```
grx_text("S.Ha.R.K.", 5, 120, 15, 0);
```

O hardware utilizado para este experimento, possui arquitetura X86, com processador AMD-Duron 950 MHz, 128 MB de memória RAM, placa mãe de 100 MHz, 512 Kb de cache, placa de

vídeo on-board 3D-NOW com 8 MB de memória de vídeo compartilhada.

Executou-se cada uma das tarefas isoladamente, garantindo-se assim que não haveria nenhuma concorrência com outra tarefa, a fim de determinar o seu tempo de computação. Através de funções de estimativas de tempo de execução, da biblioteca "Kern.h" disponível no S.Ha.R.K., obteve-se os seguintes tempos de computação: tarefa A – 8 μs, tarefa B - 21 μs e tarefa C - 21 μs.

Der acordo com a equação (1), apresentado no item 2, um conjunto com três tarefas, para que seja escalonável pelo algoritmo RM, deverá ter uma taxa de utilização do processador menor ou igual a 0.7798, como pode ser visto no cálculo abaixo:

$$3\left(\sqrt[3]{2}-1\right)=0,7798$$

Para que este mesmo conjunto de tarefas seja escalonável pelo EDF, basta ter uma taxa de utilização menor do que 1, de acordo com a equação (3).

A idéia do experimento é atribuir valores de periodicidade às três tarefas deste conjunto, de forma que tenham uma taxa de utilização do processador entre 0,7798 e 1, fazendo assim com que seja escalonável pelo EDF e não escalonável pelo RM, e em seguida averiguar qual o comportamento do S.Ha.R.K. ao escalonar o conjunto de tarefas.

Os valores de periodicidade atribuídos às tarefas foram os seguintes: tarefa A – 50 μs, tarefa B – 60 μs e tarefa C – 60 μs.

Sendo assim, a taxa de utilização do processador será de $\sum \frac{C_i}{P_i} = \frac{8}{50} + \frac{21}{60} + \frac{21}{60} = 0,86$

Portanto o conjunto de tarefas deverá ser escalonável pelo EDF, já que sua taxa de utilização do processador (0,86) é menor do que 1 (um), e não escalonável pelo RM, pois a sua taxa de utilização é maior do que o fator de escalonamento para um conjunto de 3 (três) tarefas (0,7798). Ao executar o conjunto de tarefas pelo algoritmo EDF, não houve nenhuma interrupção ou abortamento da aplicação por parte do Kernel do S.Ha.R.K. Porém, o mesmo não ocorreu quando escalonado pelo RM, em que a aplicação foi abortada já que o algoritmo não suportou escalonar este conjunto de tarefas, gerando a interrupção "ABORT 64" (Conjunto de tarefas não suportado pelo escalonador).

No segundo experimento, um conjunto com 5 (cinco) tarefas é considerado, todas independentes, e com deadline igual ao período. Todas as tarefas possuem a mesma finalidade: movimentar uma bola pelo monitor, dentro de um espaço pré-determinado. Os códigos das tarefas podem ser visualizados abaixo:

```
if ( (x > 142) || (x < 13) ) dirhor = -dirhor;        /*limites da direita e esquera*/
if ( (y > 192) || (y < 13) ) dirver = -dirver;        /*limites inferior e superior */
grx_disc(x, y, 5, 0);                                 /* desenha uma bola preta na posição anterior*/
grx_disc(x+=dirhor, y+=dirver, 5, 14);  /* desenha uma bola na nova posição */
task_endcycle();                                      /* fim da ativação */
```

Considerando o mesmo hardware especificado para o experimento anterior, executou-se cada uma das cinco tarefas isoladamente a fim de se estabelecer o seu tempo de computação. Por se tratar de tarefas idênticas, obteve-se o mesmo tempo de computação para todas as cinco tarefas: 28 μs.

Novamente realizando o cálculo da equação (1), apresentado no item 2, para que um conjunto com cinco tarefas seja escalonável, é necessário que sua taxa de utilização do processador seja menor ou igual a 0.7435, como pode ser visto no cálculo abaixo:

$$5\left(\sqrt[5]{2}-1\right)=0,7435$$

Foram atribuídos valores de periodicidade às tarefas de forma que o conjunto de tarefas tenha uma taxa de utilização do processador entre 0,7435 e 1, tornando o conjunto escalonável para o algoritmo EDF e não escalonável para o algoritmo RM.

Os valores de periodicidade atribuídos às tarefas foram os seguintes:
tarefa A – 125 μs, tarefa B – 200 μs, tarefa C – 250 μs, tarefa D – 140 μs, tarefa E – 112 μs.
Sendo assim, a taxa de utilização do processador será de:

$$\sum \frac{C_i}{P_i}=\frac{28}{125}+\frac{28}{200}+\frac{28}{250}+\frac{28}{140}+\frac{28}{112}=0,926$$

Ao executar este conjunto de tarefas, este comportou-se exatamente como o conjunto de tarefas do primeiro experimento, ou seja, foi escalonado sem nenhum abortamento pelo algoritmo EDF, já que sua taxa de utilização do processador (0,926) é menor do que 1, e não escalonado pelo RM, já que sua taxa de utilização do processador é maior do que o fator de escalonamento para um conjunto de cinco tarefas (0,7435). Neste segundo experimento, assim como no primeiro, ocorreu a interrupção “ABORT 64”.

5 Conclusão

Neste artigo foram descritos dois experimentos para teste da escalonabilidade do Kernel de Tempo Real S.Ha.R.K.. Dois conjuntos de tarefas foram submetidos ao escalonador do S.Ha.R.K., sendo que estes, teoricamente, são escalonáveis pelo algoritmo EDF e não escalonáveis pelo algoritmo RM. Os resultados obtidos comprovam que o escalonador do S.Ha.R.K. comportou-se de fato, como na teoria. Para que seja realizado um estudo mais aprofundado, torna-se necessário considerar outros fatores como o TICK (periodicidade da tarefa escalonadora) da tarefa escalonadora, além de fatores que causam interferência do sistema como um todo, a exemplo do pipeline, memória cache, entre outros. Porém o fator considerado é suficiente para observar o comportamento do Kernel do S.Ha.R.K. no experimento proposto.

Como trabalhos futuros, planeja-se realizar um estudo do comportamento do Kernel do S.Ha.R.K. ao escalonar conjuntos de tarefas que possuem taxa de utilização do processador próximos ao fator de cálculo do algoritmo RM, considerando ainda o processo da tarefa escalonadora, a fim possibilitar uma melhor avaliação da eficiência do escalonador.

Referências Bibliográficas

ABENI, L.; LAMASTRA, G. The OSLib Project. Pisa, 1998. Disponível em :<http://oslib.sssup.it>.
acesso em: 05 fev. 1999.

BURNS, A.; WELLINGS, A. Real-Time Systems and Programming Languages, Second edition. Addison-Wesley,1997.

FARINES, J.; FRAGA, J.; OLIVEIRA, R. S.“Sistemas de Tempo Real”, Florianópolis, 2000. 193p.

GAI, P. S.Ha.R.K. User Manual. Pisa, 2001. 193p.

JONES, M, “What really happened on Mars?”, Risks Forum, RISKS-19.49, Dec 1997.

LIU, C. L.; LAYLAND, J.W. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. Journal of the ACM, Vol. 20, No. 1, pp. 46-61, january 1973.

PINEDO, M. Scheduling: Theory, Algorithms and Systems. Prentice-Hall, 1995.

RAMAMRITHAN, J.; STANKOVIC, A. Schedulung Algorithms and Operating Systems Support for RealTime Systems. Proccedings of the IEEE, Vol. 82, Nº 1, January 1994.