Análise Temporal da Implementação da Fila de Aptos como Lista Ordenada e Lista Desordenada

Ricardo Bacha Borges, Rômulo Silva de Oliveira

Departamento de Automação e Sistemas (DAS)
Universidade Federal de Santa Catarina (UFSC)
Caixa Postal 476 – 88040-900 – Florianópolis – SC – Brasil
{rbb,romulo}@das.ufsc.br

Abstract. *The ready queue maintains the tasks that are waiting to execute. There are many ways to implement the ready queue, but the classic form consists of using a chained list sorted according to the task priorities. An alternative, although rarely used, would be to maintain an unsorted chained list. The objective of this paper is to make a timing analysis of the implementation of the ready queue. The same analysis is accomplished for the implementations as sorted list and unsorted list. The analysis shows that, for real-time systems, it may be better to maintain the ready queue unsorted.*

Resumo. *A fila de aptos (ready queue) mantém as tarefas que estão esperando para executar. Existem muitas formas de implementar a fila de aptos, mas a forma clássica consiste em usar uma lista encadeada ordenada conforme a prioridade de execução das tarefas. Uma implementação alternativa, porém raramente utilizada, seria manter a lista encadeada desordenada. O objetivo deste artigo é realizar uma análise temporal da implementação da fila de aptos. A mesma análise é realizada para as implementações como lista ordenada e como lista desordenada. A análise mostra que, para sistemas de tempo real, pode ser melhor manter a fila de aptos desordenada.*

1. Introdução

Sistemas de tempo real são aqueles caracterizados pela existência de requisitos de natureza temporal explícitos e não triviais em sua especificação. Nesse tipo de sistema, os aspectos temporais não estão limitados a questões de desempenho, mas sim à própria utilidade do sistema. Um sistema de tempo real que não atende aos seus requisitos temporais simplesmente não funciona. Enquanto nos sistemas convencionais o objetivo do projetista é executar as tarefas o mais rápido possível, em sistemas de tempo real o objetivo do projetista é executar as tarefas no tempo correto, ou seja, no tempo especificado.

A forma mais comum de explicitar requisito temporal é estabelecer deadlines para certas tarefas. É necessário garantir em tempo de projeto que o tempo de resposta da tarefa em questão será menor ou igual ao seu deadline, isto no pior cenário possível. O não atendimento a este requisito pode resultar em catástrofes no caso dos sistemas críticos (*hard deadline*) ou na inutilidade do sistema desenvolvido no caso dos sistemas críticos à missão (*firm deadline*).

Assim como aplicações convencionais, aplicações de tempo real são mais facilmente construídas se puderem aproveitar os serviços de um sistema operacional (SO). Desta forma, o programador da aplicação não precisa preocupar-se com a gerência dos recursos básicos (processador, memória física, controladores de periféricos). Entretanto, uma vez que tanto a aplicação como o SO compartilham os mesmos recursos do hardware, o comportamento temporal do SO afeta o comportamento temporal da aplicação. Por exemplo, considere a rotina do sistema operacional que trata as interrupções do temporizador em hardware. O projetista da aplicação pode ignorar completamente a função desta rotina, mas não pode ignorar o seu efeito temporal, isto é, a interferência que ela causa nas tarefas da aplicação.

No caso dos sistemas embutidos (*embedded system*) de pequeno porte com requisitos temporais, as limitações de memória, de energia e de capacidade de processamento exigem a utilização de sistemas operacionais de propósito específico, ajustados exatamente para a aplicação em questão. Limitados a um simples microkernel, esses sistemas operacionais são construídos pelo próprio desenvolvedor da aplicação ou, pelo menos, alterados e configurados por este.

Uma estrutura de dados quase sempre presente nos sistemas operacionais de qualquer porte é a fila de aptos (*ready queue*). Ela mantém a fila das tarefas que estão esperando para executar, ou seja, esperando para que o seu contexto de execução seja carregado e o processador passe a executar as instruções da sua aplicação. Existem muitas formas de implementar a fila de aptos, mas a forma clássica consiste em usar uma lista encadeada unindo os descritores das várias tarefas. Tipicamente, esta lista encadeada é mantida ordenada conforme a prioridade de execução das tarefas. A liberação de uma tarefa implica em uma inserção na lista encadeada ordenada, enquanto a localização da próxima tarefa a executar é a primeira posição da fila. Uma implementação alternativa, porém raramente utilizada, seria manter a lista encadeada desordenada. Neste caso, a liberação de uma tarefa implica em uma inserção em qualquer lugar da lista. Já a seleção da próxima tarefa a executar implica em percorrer toda a lista encadeada desordenada para localizar a tarefa com maior prioridade.

O objetivo deste artigo é realizar uma análise temporal da implementação da fila de aptos. A mesma análise é realizada para as implementações como lista ordenada e como lista desordenada. A análise mostra que, para sistemas de tempo real, pode ser melhor manter a fila de aptos desordenada. Este é um resultado não intuitivo, possível através de uma modelagem algébrica do comportamento do sistema no tempo.

Na seção 2 são listados vários trabalhos presentes na literatura que tratam da implementação da fila de aptos em sistemas de tempo real. Na seção 3 é estabelecida a arquitetura de software considerada neste artigo. A seção 4 contém a análise temporal realizada sobre as implementações clássicas da fila de aptos. A seção 5 contém um exemplo numérico da análise realizada. Finalmente, os comentários finais aparecem na seção 6.

2. Trabalhos Relacionados

Tradicionalmente são utilizados algoritmos e estruturas de dados convencionais, tais como listas encadeadas, na implementação das filas dentro dos sistemas operacionais. O processamento de listas encadeadas introduz substancial *overhead* em cenários de pior

caso, como na liberação simultânea de várias tarefas periódicas. A complexidade computacional desta operação está associada com $O(n^2)$, pois no pior caso é necessário mover n tarefas da lista encadeada de espera para a lista encadeada que representa a fila do processador [Katcher;Arakawa;Strosnider 1993]. Em [Burns;Tindell;Wellings 1995] é mostrado experimentalmente que a liberação simultânea de 20 tarefas periódicas pode ocupar até 570us do tempo de um processador M68020 executando o sistema operacional Olympus. Nesse mesmo processador e sistema operacional a liberação de uma única tarefa demora apenas 76us.

Segundo [Angelov;Ivanov;Burns 2002], o problema do *overhead* associado com a liberação de tarefas pode ser minimizado através da substituição da tradicional implementação da fila do processador como uma lista encadeada. No lugar, a fila do processador pode ser pensada como um conjunto de vetores booleanos, resultando em operações rápidas e tempo de execução constante, independente do número de tarefas envolvidas.

Em [Brown 1988] e [Mhatre 2001] são apresentados estudos interessantes sobre a implementação de filas de prioridade no contexto dos sistemas de tempo real. Em [Leyva-Del-Foyo;Mejia-Alvares;Niz 2006] os autores descrevem uma técnica de implementação na qual as interrupções de hardware respeitam as prioridades das interrupções de software, resultando em um sistema mais previsível. Entretanto, no caso das interrupções do *timer* a técnica apresenta limitações pois a prioridade da interrupção é variável, isto é, depende das tarefas que precisam ser acordadas. Em [Oliveira 2003] são discutidos aspectos construtivos dos sistemas operacionais de tempo real, inclusive a implementação da fila de aptos, mas nenhuma modelagem algébrica é apresentada.

3. Arquitetura de Software Considerada

Neste trabalho é suposto que cada tarefa do sistema é representada por um descritor, o qual inclui apontadores para a montagem de uma lista. A fila de aptos é implementada como uma lista ordenada ou desordenada, onde os próprios descritores de tarefa são utilizados como elementos da lista.

Como é típico em sistemas de tempo real, será suposto o emprego de escalonamento preemptivo baseado em prioridades fixas. O SO não altera as prioridades das tarefas definidas pelo projetista da aplicação. Este mecanismo permite que o projetista da aplicação defina as prioridades das tarefas utilizando políticas bem conhecidas, como o taxa monotônica (*rate monotonic*) e o deadline monotônico (*deadline monotonic*) [Audsley;Burns;Wellings 1993].

Neste trabalho também é suposto que as tarefas são periódicas ou esporádicas. Tarefas periódicas possuem um intervalo de tempo fixo entre liberações consecutivas (período). Tarefas esporádicas apresentam um intervalo mínimo de tempo entre cada duas liberações consecutivas. No caso das tarefas periódicas, a espera até o próximo instante de liberação é feita através de uma função de temporização (*sleep*). O tempo de espera é o instante da próxima liberação menos o instante de solicitação da espera.

A implementação da função de temporização (*sleep*) utiliza uma lista encadeada de descritores de espera. Esta lista é mantida ordenada conforme o instante de acordar. Desta forma, a primeira tarefa a ser acordada é indicada pelo primeiro descritor desta lista. É suposta a existência de um temporizador em hardware (*timer*) que suporta esta

funcionalidade. Como é típico em sistemas de tempo real, o temporizador em hardware não opera em "modo periódico" mas sim em "modo de disparo único" (*single-shot*). O temporizador em hardware é programado para gerar uma interrupção apenas no próximo instante de interesse. Uma vez que a próxima tarefa a ser acordada é a primeira da lista, o temporizador em hardware é sempre programado conforme o tempo de espera desta tarefa. Após o seu disparo, ele é reprogramado para acordar a tarefa seguinte da lista, e assim sucessivamente. Na literatura de tempo real este mecanismo é conhecido como temporizador de alta resolução.

Neste trabalho será considerada a existência de N tarefas de software no sistema, incluindo em N tanto as tarefas da aplicação como quaisquer tarefas de sistema existentes (*threads de kernel*). Cada tarefa Ti, com i entre 1 e N, é identificada neste trabalho pela sua prioridade i. A tarefa T_1 é a tarefa mais prioritária do sistema, enquanto a tarefa T_N é a menos prioritária. Todas as tarefas são periódicas ou esporádicas. Além das tarefas de software, é preciso considerar o tratador das interrupções geradas pelo temporizador em hardware, conforme a implementação das temporizações.

4. Análise Temporal

Esta seção apresenta a análise temporal de pior caso para a implementação da fila de aptos como lista ordenada e como lista desordenada. Inicialmente é descrito o método de análise empregado neste estudo.

4.1. Método de Análise Temporal

Neste trabalho são usados os métodos de análise de escalonabilidade desenvolvidos por Audsley em sua tese [Audsley 1994], para tarefas com deadline menor ou igual ao período, os quais estendem o trabalho anterior de Joseph e Pandya [Joseph;Pandya 1989]. Posteriormente, Tindell em sua tese [Tindell 1994] estendeu este tipo de análise para tarefas com deadline arbitrário, entre outras coisas. Descrições da abordagem podem ser encontradas em [Audsley et al. 1993], [Audsley;Burns;Wellings 1993], [Tindell;Burns;Wellings 1994], [Burns;Tindell;Wellings 1995]. Embora esta abordagem também tenha sido estendida para ambientes distribuídos, este artigo trata exclusivamente de sistemas com um único processador.

De forma simplificada, a análise de escalonabilidade em questão assume que uma prioridade fixa é associada a cada tarefa em tempo de projeto. Durante a execução é utilizado um escalonador preemptivo baseado em prioridades. Testes de escalonabilidade em tempo de projeto calculam o tempo máximo de resposta para cada tarefa e os comparam com os respectivos deadlines, avaliando dessa forma a escalonabilidade do sistema. Os testes de escalonabilidade verificam um conjunto de tarefas com prioridades fixas arbitrárias, atribuídas de acordo com qualquer política. Em muitos casos, a atribuição de prioridades pelo deadline monotônico é ótima [Leung;Whitehead 1982].

A limitação de espaço impede uma descrição detalhada das equações empregadas na análise de escalonabilidade usada neste trabalho. A título de ilustração, considere um sistema formado por N tarefas, onde cada tarefa Ti é periódica com período Pi ou esporádica com intervalo mínimo entre ativações Pi. Cada tarefa Ti possui uma prioridade fixa $\rho(T_i)$. Cada tarefa Ti também possui um tempo máximo de computação

Ci, um deadline Di e uma variação na liberação máxima (*release jitter*) Ji. É suposto que Di≤Pi para todas as tarefas. Em cada ativação da tarefa Ti ela pode ser bloqueada por tarefas de menor prioridade no máximo Bi unidades de tempo.

Dadas as condições acima, o tempo máximo de resposta Ri de cada tarefa Ti pode ser calculado através das equações abaixo, onde Wi é obtido através de um processo iterativo, com complexidade pseudo-polinomial. O conjunto HP(i) é formado por todas as tarefas com prioridade maior ou igual a prioridade da tarefa Ti.

$$W_i = C_i + B_i + \sum_{j \in HP(i)} \left\lceil \frac{J_j + W_i}{P_j} \right\rceil \times C_j \qquad \text{[Eq. 1]}$$

$$R_i = J_i + W_i \qquad \text{[Eq. 2]}$$

As equações apresentadas continuam válidas quando políticas de gerência de recursos tais como herança de prioridade e *priority ceiling* são usadas nas situações de exclusão mútua. Por outro lado, as equações apresentadas são válidas apenas para deadline menor ou igual ao período. Equações semelhantes são apresentadas em [Tindell;Burns;Wellings 1994] para deadlines arbitrários (podem ser maiores que o período).

4.2. Modelagem da Arquitetura Considerada

O objetivo desta seção é modelar algebricamente o atraso que uma tarefa Ti sofre em função das operações relacionadas com a fila de aptos. Obviamente este atraso está associado com operações sobre a fila de aptos tanto relacionadas com a própria tarefa Ti como relacionadas com outras tarefas, em função das interrupções do temporizador em hardware. O primeiro passo na modelagem é definir pseudo-tarefas que representem os fluxos de execução que existem além das tarefas da aplicação. No caso deste estudo, as execuções do tratador de interrupções do *timer* precisam ser modeladas.

Quando o temporizador em hardware gera interrupções periódicas a modelagem é simples, basta definir uma pseudo-tarefa com o período do *timer* e o tempo de execução no pior caso (*worst-case execution time*) do tratador de interrupções do *timer*. Esta abordagem simples não é possível neste estudo, uma vez que o temporizador em hardware é sucessivamente programado no modo “disparo único”.

Para modelar o comportamento do tratador de interrupções do timer deve-se observar que ele é ativado sempre que inicia o período de alguma tarefa de software. Ou seja, cada tarefa da aplicação dá origem a uma pseudo-tarefa, associada com a execução do tratador de interrupções do *timer* quando ele acorda a tarefa da aplicação em questão.

No caso do sistema estudado, para cada Ti, 1≤i≤N, existe uma pseudo-tarefa Hi, cujo período P(Hi) é o mesmo período da tarefa Ti. Entretanto, a prioridade de qualquer tarefa Hi será sempre maior que a prioridade de qualquer tarefa Ti, uma vez que Hi representa a execução do tratador de interrupções do *timer*, a qual acontece mesmo quando T1 está executando. Em resumo, qualquer tarefa Hk, 1≤k≤N, interfere com todas as tarefas Ti, 1≤i≤N.

Outra questão relevante é o número de vezes que Hk pode ocorrer durante a execução de Ti. Uma vez que foi assumido Di≤Pi, temos que uma nova liberação da tarefa Ti só vai ocorrer após a liberação anterior da tarefa Ti estar concluída. Ou seja,

para cada tarefa Ti, ou ela está na fila de aptos esperando pelo processador, ou ela está na fila de *sleep* esperando pela interrupção do *timer*.

O tempo de execução no pior caso C(Hi) é o tempo que o tratador de interrupções do *timer* demora, no pior caso, quando a tarefa Ti é acordada. O valor deste tempo é discutido nas seções seguintes.

Vamos agora adaptar as equações [Eq.1] e [Eq.2] para o cenário deste estudo. O termo Bi surge de situações de exclusão mútua (sincronização entre tarefas) quando uma tarefa de mais baixa prioridade executa uma operação de *lock* sobre uma variável tipo *mutex* ou semáforo. Isto pode bloquear a execução de outra tarefa de mais alta prioridade, quando no futura ela executa uma operação *lock* sobre a mesma variável de sincronização. Considerando o objetivo deste estudo, vamos considerar Bi=0 para todas as tarefas, ou seja, é suposto que as situações de exclusão mútua da aplicação afetam os tempos de resposta de forma similar, não importando como a fila de aptos foi implementada.

Por outro lado, toda execução da tarefa Ti deve necessariamente ser precedida por uma execução de Hi, ou seja, a execução do tratador da interrupção de timer que exatamente libera a tarefa Ti. Isto pode ser modelado como Ji = C(Hi). O tempo de computação de Hi representa o atraso para a liberação de Ti.

Qualquer período de interrupções desabilitadas representa um possível atraso de liberação para os tratadores de interrupções. Este valor será definido pelo tempo máximo de manipulação das filas de aptos e de espera, pois as mesmas exigem exclusão mútua entre os tratadores de interrupção e, portanto, interrupções desabilitadas. Desta forma, temos que J(H1)=J(H2)=J(H3)=J(H4)=J(H5)=JH, que depende da forma de implementação da fila de aptos.

Quando uma tarefa da aplicação conclui sua execução, o processador fica livre e é necessário selecionar a próxima tarefa para execução. O tempo necessário para selecionar, na fila de aptos, qual tarefa será executada a seguir, também deve ser somado ao tempo de resposta das tarefas de mais baixa prioridade. Pode-se modelar este efeito substituindo Cj por CAj na [Eq. 1], ou seja, CAj representa o tempo de execução no pior caso da tarefa Tj acrescido pelo tempo necessário para escolher, na fila de aptos, o seu sucessor. O valor do ajuste depende da forma como a fila de aptos é implementada.

Com respeito às interferências, cada tarefa Ti será afetada por Hk da seguinte forma: possivelmente várias vezes se k < i (Tk executa várias vezes enquanto Ti executa uma), nenhuma vez se k = i (o efeito de Hk está modelado pelo Ji) ou no máximo uma única vez se k > i (Tk não pode executar antes de Ti terminar).

Pode-se reescrever as equações do tempo de resposta em função dessas colocações, definindo ainda PHP(i) (Pseudo Higher Priority) como o conjunto de tarefas {Hk | k < i} e PLP(i) (Pseudo Lower Priority) como o conjunto de tarefas {Hk | k > i}.

$$W_i = C_i + \sum_{j \in HP(i)} \left\lceil \frac{J_j + W_i}{P_j} \right\rceil \times CA_j + \sum_{k \in PHP(i)} \left\lceil \frac{J_H + W_i}{P_{Hk}} \right\rceil \times C_{Hk} + \sum_{k \in PLP(i)} C_{Hk}$$

[Eq. 3]

$$R_i = J_i + W_i \qquad \text{[Eq. 4]}$$

4.3. Fila de Aptos como Lista Ordenada

Quando a pseudo-tarefa Hi executa, a primeira tarefa da fila de espera é a tarefa Ti, pela própria definição de Hi. O tratador da interrupção do *timer* deve então remover a primeira tarefa da fila de espera, cujo tempo de execução é "remove_p". Em seguida, o tratador de interrupções do *timer* deve inserir Ti na fila de aptos. Como a fila de aptos é mantida como uma lista ordenada, o tempo de inserção depende do tamanho atual da fila de aptos e da prioridade da tarefa sendo inserida. No pior caso, a tarefa sendo inserida possui uma prioridade mais baixa do que todas as demais tarefas presentes na fila de aptos. Isto exige um caminhamento seqüencial sobre a lista encadeada até o seu final. Por exemplo, a Tabela 1 mostra os valores para C(Hi) quando N=5, insere(x) representa o tempo de inserção quando a fila tem tamanho x.

Tabela 1. Valores de C(Hi) quando N=5, lista ordenada

Pseudo-tarefa	Remoção da fila de espera	Inserção na fila de aptos
H1 (acordar T1)	remove_p	insere(0)
H2 (acordar T2)	remove_p	insere(1)
H3 (acordar T3)	remove_p	insere(2)
H4 (acordar T4)	remove_p	insere(3)
H5 (acordar T5)	remove_p	insere(4)

A Tabela 2 mostra os valores característicos das pseudo-tarefas quando N=5. Com respeito às prioridades, temos que:

$\rho(H1) = \rho(H2) = \rho(H3) = \rho(H4) = \rho(H5) > \rho(T1) > \rho(T2) > \rho(T3) > \rho(T4) > \rho(T5)$.

Tabela 2. Valores característicos das peudo-tarefas, lista ordenada

Pseudo-tarefa	Período	Tempo de execução	Prioridade
H1	P(H1) = P1	C(H1)=remove_p+insere(0)	ρ(H1)
H2	P(H2) = P2	C(H2)=remove_p+insere(1)	ρ(H2)
H3	P(H3) = P3	C(H3)=remove_p+insere(2)	ρ(H3)
H4	P(H4) = P4	C(H4)=remove_p+insere(3)	ρ(H4)
H5	P(H5) = P5	C(H5)=remove_p+insere(4)	ρ(H5)

Quando uma tarefa da aplicação conclui sua execução, o processador fica livre e é necessário selecionar qual será a próxima tarefa a executar. Como a lista está ordenada, basta pegar a primeira tarefa da lista para execução. Este tempo é semelhante ao tempo para remover uma tarefa da fila de espera, ou seja, "remove_p". Logo, temos neste caso que CAj = Cj + remove_p.

Neste tipo de implementação, no pior caso as interrupções poderão ficar desabilitadas por remove_p+insere(N-1) unidades de tempo. Este deve ser o valor usado como variação de liberação (*release jitter*) no caso das pseudo-tarefas Hi, ou seja:

JH = remove_p+insere(N-1).

4.4. Fila de Aptos como Lista Desordenada

Novamente, quando a pseudo-tarefa Hi executa, a primeira tarefa da fila de espera é a tarefa Ti, pela própria definição de Hi. O tratador da interrupção do *timer* deve então remover a primeira tarefa da fila de espera, cujo tempo de execução é "remove_p". Em seguida, o tratador de interrupções do *timer* deve comparar a prioridade da tarefa sendo acordada Ti com a prioridade da tarefa executando. Caso a prioridade de Ti seja mais alta, ocorre uma troca de contexto e a tarefa que estava executando é inserida em qualquer lugar da fila de aptos. Caso Ti não seja a tarefa mais prioritária, ela é inserida em qualquer lugar da fila de aptos. De qualquer forma, teremos uma inserção em lista encadeada desordenada, cujo tempo de execução é "insere_p".

Por exemplo, a Tabela 3 mostra os valores para C(Hi) quando N=5. A Tabela 4 mostra os valores característicos das pseudo-tarefas quando N=5. Com respeito às prioridades, temos novamente que:

$\rho(H1) = \rho(H2) = \rho(H3) = \rho(H4) = \rho(H5) > \rho(T1) > \rho(T2) > \rho(T3) > \rho(T4) > \rho(T5)$.

Tabela 3. Valores de C(Hi) quando N=5, lista desordenada

Pseudo-tarefa	Remoção da fila de espera	Inserção na fila de aptos
H1 (acordar T1)	remove_p	insere_p
H2 (acordar T2)	remove_p	insere_p
H3 (acordar T3)	remove_p	insere_p
H4 (acordar T4)	remove_p	insere_p
H5 (acordar T5)	remove_p	insere_p

Tabela 4. Valores característicos das peudo-tarefas, lista desordenada

Pseudo-tarefa	Período	Tempo de execução	Prioridade
H1	P(H1) = P1	C(H1)=remove_p +insere_p	ρ(H1)
H2	P(H2) = P2	C(H2)=remove_p +insere_p	ρ(H2)
H3	P(H3) = P3	C(H3)=remove_p +insere_p	ρ(H3)
H4	P(H4) = P4	C(H4)=remove_p +insere_p	ρ(H4)
H5	P(H5) = P5	C(H5)=remove_p +insere_p	ρ(H5)

Também é necessário selecionar qual será a próxima tarefa a executar, quando uma tarefa da aplicação conclui sua execução. Como a lista está desordenada, será necessário percorrer toda a lista desordenada e selecionar a tarefa na fila de aptos com a maior prioridade. Sabe-se no entanto que, quando a tarefa Tj conclui, todos os membros da lista possuem prioridade menor que j. No pior caso, quando Tj conclui, a fila de aptos inclui todas as tarefas de j+1 até N. Logo, temos que CAj = Cj + remove(N-j), onde remove(x) representa o tempo para localizar e remover a tarefa mais prioritária em uma lista desordenada de tamanho x.

Neste tipo de implementação, no pior caso, as interrupções poderão ficar desabilitadas por remove(N-1) unidades de tempo. Este deve ser o valor usado como variação de liberação (*release jitter*) no caso das pseudo-tarefas Hi, ou seja:

JH = remove(N-1).

4.5. Comentários

Quando a fila de aptos é implementada por uma lista ordenada, a operação de escolha da próxima tarefa é rápida, basta utilizar a primeira tarefa da lista. Entretanto, esta solução traz uma desvantagem. Enquanto uma tarefa mais prioritária está executando, interrupções do *timer* que sinalizam a liberação de tarefas menos prioritárias são tratadas. Isto significa parar temporariamente a execução da tarefa mais prioritária para retirar da fila de espera e inserir corretamente na fila de aptos uma tarefa que será executada no futuro. Temos na verdade uma inversão de prioridades, modelada como a interferência de, por exemplo, H5 sobre T1.

A solução baseada em lista desordenada minimiza este problema ao simplificar o atendimento a esta interrupção. A inserção desordenada tem um custo menor que a inserção oredenada, reduzindo a interferência sofrida pela tarefa de mais alta prioridade. Entretanto, o preço a ser pago é um maior tempo de chaveamente de contexto, pois agora é sempre necessário percorrer toda a fila de aptos para decidir qual a próxima tarefa a executar. Este atraso afeta principalmente as tarefas de mais baixa prioridade.

Os efeitos quantitativos da escolha do tipo de lista dependem em parte do número de tarefas e das relações entre os períodos das mesmas. A próxima seção traz um exemplo numérico onde a modelagem algébrica construida neste artigo é aplicada.

5. Exemplo Numérico

Para ilustrar os modelos construídos na seção anterior, vamos considerar uma aplicação hipotética. No sentido de usar números realistas, foram implementadas listas encadeadas em um DSP (*Digital Signal Processor*) e os tempos de operação foram medidos. Obviamente os tempos exatos dependem do modelo do processador, do código fonte e do compilador usado. Os números apresentados aqui foram medidos em um DSP 2407 da Texas Instruments. As rotinas foram programadas em C e compiladas na ferramenta “Code Composer Studio”, também da Texas Instruments.

No exemplo numérico serão utilizados os tempos para manipulação de listas mostrados na Tabela 5. O valor x indica o tamanho da lista antes da operação.

Tabela 5. Valores numéricos para operações sobre listas

Operação	Tempo em microsegundos
Insere_p	0.9
Insere(x)	$0.9 + 0.6 \times x$
Remove_p	0.7
Remove(x)	$0.7 + 0.7 \times x$

A Tabela 6 mostra os parâmetros que caracterizam as tarefas da aplicação.

Tabela 6. Tarefas da aplicação exemplo

Tarefa	Período (uS)	Tempo de execução (uS)
T1	50	4
T2	50	10
T3	300	30
T4	500	50
T5	500	50

A tabela 7 mostra como ficam as pseudo-tarefas no caso da lista ordenada. Temos ainda que, para lista ordenada, CAj = Cj + 0.7 e JH = 0.7+3.3 = 4.0.

Tabela 7. Valores característicos das peudo-tarefas, lista ordenada

Pseudo-tarefa	Período (uS)	Tempo de execução (uS)
H1	P(H1) = 50	C(H1) = 0.7 + 0.9 = 1.6
H2	P(H2) = 50	C(H2) = 0.7 + 1.5 = 2.2
H3	P(H3) = 300	C(H3) = 0.7 + 2.1 = 2.8
H4	P(H4) = 500	C(H4) = 0.7 + 2.7 = 3.4
H5	P(H5) = 500	C(H5) = 0.7 + 3.3 = 4.0

A tabela 8 mostra como ficam as pseudo-tarefas no caso da lista desordenada. Temos ainda que, para lista desordenada: $CAj = Cj + 0.7 + 0.7 \times (5-j)$ e JH = remove(4), ou seja, JH = 3.5.

Tabela 8. Valores característicos das peudo-tarefas, lista desordenada

Pseudo-tarefa	Período (uS)	Tempo de execução (uS)
H1	P(H1) = 50	C(H1)= 0.7+0.9=1.6
H2	P(H2) = 50	C(H2)= 0.7+0.9=1.6
H3	P(H3) = 300	C(H3)= 0.7+0.9=1.6
H4	P(H4) = 500	C(H4)= 0.7+0.9=1.6
H5	P(H5) = 500	C(H5)= 0.7+0.9=1.6

Finalmente, a Tabela 9 mostra os tempos de resposta no pior caso de todas as cinco tarefas da aplicação. Como esperado, o tempo de resposta das tarefas de maior prioridade diminui, enquanto o tempo de resposta das tarefas de menor prioridade aumentou. Isto mostra que, do ponto de vista dos sistemas de tempo real, pode ser vantajoso implementar a fila de aptos como uma lista desordenada.

Tabela 9. Tempos de resposta das tarefas da aplicação

Tarefa	Tempo de resposta (uS) Lista Ordenada	Tempo de resposta (uS) Lista Desordenada
T1	18.0	12.0
T2	28.7	25.5
T3	78.6	81.8
T4	148.5	180.9
T5	237.6	279.3

6. Conclusões

Neste artigo foi apresentada a modelagem temporal da implementação da fila de aptos como lista ordenada e como lista desordenada. Foram construídas equações que permitem calcular o tempo de resposta das tarefas da aplicação a partir das características básicas do sistema.

A modelagem mostrou que manter a lista desordenada diminui o tempo de resposta no pior caso das tarefas de mais alta prioridade. O preço a ser pago é um maior tempo de resposta para as tarefas de prioridade mais baixa. Considerando-se que em sistemas de tempo real o mais importante é garantir o cumprimento dos deadlines por parte das tarefas, o uso de uma lista desordenada para implementar a fila de aptos poderá viabilizar o atendimento do deadline da tarefa mais prioritária, quando a fila de aptos como lista ordenada não conseguiria. Esta questão torna-se tão mais importante quanto menores forem os tempos de execução das tarefas, ou seja, mais relevante o *overhead* imposto pelo sistema operacional. No caso de sistemas exigentes, a fila de aptos pode ser implementada com estruturas de dados mais complexas, como discutido na seção 2.

Na literatura de tempo real existe teoria suficiente para a modelagem completa de um sistema operacional. Embora seja difícil aplicar a modelagem em sistemas

operacionais antigos, construídos sem preocupações de tempo real, é viável aplicá-la a um sistema operacional completo, desde que ele seja construido com algum cuidado. É razoável esperar que, em alguns anos, existirão sistemas operacionais completos (com funcionalidade semelhante ao Linux e ao Windows XP), onde equações permitirão o cálculo do tempo máximo de resposta para cada tarefa do sistema, mesmo que tarefas utilizem livremente os serviços do sistema operacional.

References

Angelov, C. K., Ivanov, I. E., Burns, A. (2002) "HARTEX - a safe real-time kernel for distributed computer control systems" Software - Practice and Experience 32(3), pages 209-232.

Audsley, N. C., Burns, A., Richardson, M. F., Tindell, K., Wellings, A. J. (1993) "Applying New Scheduling Theory to Static Priority Pre-emptive Scheduling" Software Engineering Journal, Vol. 8, No. 5, pages 284-292.

Audsley, N. C., Burns, A., Wellings, A. J. (1993) "Deadline Monotonic Scheduling Theory and Application" Control Engineering Practice, Vol.1, No.1, pages 71-78.

Audsley, N. C. (1994) "Flexible Scheduling of Hard Real-Time Systems" Department of Computer Science Thesis, University of York.

Brown, R. (1988) "Calendar queues: a fast 0(1) priority queue implementation for the simulation event set proble" Communications of the ACM, Volume 31, Issue 10, October 1988, pages 1220-1227.

Burns, A., Tindell, K., Wellings, A. (1995) "Effective Analysis for Engineering Real-Time Fixed-Priority Schedulers" IEEE Trans. on Soft. Eng., Vol. 21, pages 475-480.

Joseph, M., Pandya, P. (1989) "Finding Response Times in a Real-Time System" BCS Computer Journal, Vol 29, N^o 5, pages 390-395.

Katcher, D., Arakawa, H., Strosnider, J. (1993) "Engineering and Analysis of Fixed-Priority Schedulers" IEEE Trans. on Soft. Eng., Vol. 19, pages 920-934.

Leung, J. Y. T., Whitehead, J. (1982) "On the Complexity of Fixed-Priority Scheduling of Periodic, Real-Time Tasks" Performance Evaluation, 2 (4), pages 237-250.

Leyva-del-Foyo, L. E., Mejia-Alvarez, P., Niz, D. de (2006) "Predictable Interrupt Management for Real-Time Kernels over Conventional PC Hardware" RTAS - IEEE Real-Time and Embedded Technology and Applications Symposium, California, United States.

Mhatre, N. (2001) "A Comparative Performance Analysis of Real-Time Priority Queues" Master Dissertation, The Florida State University.

Oliveira, R. S. de (2003) "Aspectos Construtivos dos Sistemas Operacionais de Tempo Real" WTR'2003 - 5o Workshop de Tempo Real. Natal - RN.

Tindell, K. W. (1994) "Fixed Priority Scheduling of Hard Real-Time Systems" Department of Computer Science Thesis, University of York.

Tindell, K. W., Burns, A., Wellings, A. J. (1994) "An Extendible Approach for Analyzing Fixed Priority Hard Real-Time Tasks" Real-Time Systems, pages 133-151.