Qualidade de Serviço para Programas Móveis de Tempo Real

Eliane G. Monteiro e J.C.B. Leite
Universidade Federal Fluminense, Instituto de Computação
Niterói, Brasil, Cep 24210-240
{egm, julius}@ic.uff.br

Abstract

The specification and management of quality of service (QoS) has become an important requirement for supporting the execution of programs that can migrate from one computer to another, especially for dealing with time constraints in multimedia applications. This work presents a QoS policy proposal for mobile programs, exploring the allocation of CPU cycles according to the characteristics and needs of each program. Considering that many mobile programs have real-time requirements (of the soft real-time type), a policy proposal should incorporate a scheduling scheme for these programs as well as for non-real-time programs running on the same machine. To this end, it is necessary that the scheduling scheme can be adapted to changes in the load of the machine in use without interfering with the execution of the real-time tasks. Through the implementation of the scheduling algorithm in the Java-Linux environment, this scheme is assessed

Keywords: Operating systems, Real-time scheduling, QoS policy, Mobile programs.

Resumo

A especificação e o gerenciamento da qualidade de serviço tem se tornado um fator importante para o atendimento de programas que migram de um computador para outro, em particular, para o atendimento das restrições de tempo das aplicações multimídia. Neste sentido, este trabalho apresenta uma proposta de política de QoS para programas móveis, sendo particularmente explorada a liberação de ciclos de UCP de acordo com as características e necessidades de cada programa. Considerando-se que muitos dos programas móveis possuem características de tempo real (do tipo soft real-time), a política proposta incorpora um esquema de escalonamento de tarefas que suporta a execução desses programas em conjunto com os programas convencionais (sem requisitos de tempo). Para tal, é necessário que o esquema de escalonamento se adapte às variações de carga do ambiente de execução, sem prejudicar a execução das tarefas de tempo real. Através da implementação do algoritmo de escalonamento no ambiente Java-Linux, esse esquema é avaliado.

Palavras chaves: Sistemas operacionais, Escalonamento de tempo real, Política de QoS, Programas móveis.

1 Introdução

A popularização do modelo cliente-servidor com programas migrando de um computador para outro (aqui chamados de programas móveis) e sendo executados no seu destino tem sido fonte de interesse para diversas linhas de pesquisa. A especificação e o gerenciamento da qualidade de serviço (QoS – *Quality of Service*) tem se tornado, portanto, um fator importante no atendimento destes programas móveis [5, 7, 10, 11, 13, 20]. Em particular, pode-se citar a necessidade de suporte às restrições de tempo das crescentes aplicações multimídia (como por exemplo, apresentação de vídeo, sintetizador de voz e videoconferência). A fim de atender os requisitos de tempo impostos por essas atividades de tempo real e também atender às atividades convencionais, diferentes políticas de controle de recursos de UCP vêm sendo estudadas e apresentadas. Estas políticas visam solucionar o problema do escalonamento de tarefas para ambientes mistos, onde tarefas com restrições de tempo e tarefas convencionais competem pelo mesmo recurso [12, 14, 17, 20, 23, 24, 31].

1.1 Parâmetros de QoS

De uma forma geral, em um sistema do tipo cliente-servidor, a condição de QoS começa com a definição dos parâmetros exigidos pelo cliente, segue com as especificações e limitações impostas pela tecnologia em uso e finaliza com procedimentos de supervisão e controle, responsáveis pela manutenção da qualidade do serviço em um nível aceitável. Com base nesta visão geral, e a partir das considerações de [10] os parâmetros de QoS podem ser agrupados em: (i) parâmetros de QoS baseados no cliente; (ii) parâmetros de QoS baseados em tecnologia e (iii) parâmetros de QoS baseados no gerenciamento. São exemplos de parâmetros baseados no cliente: a prioridade, o peso e a cota associada à importância do cliente; prazo final de entrega (*deadline*), a taxa de amostragem e a reserva de recurso associada à solicitação do cliente; o custo por uso e o custo por unidade associado à contabilidade do cliente, entre outros.

Os parâmetros baseados no gerenciamento consistem em funções de supervisão e controle indispensáveis à obtenção e manutenção da qualidade do serviço em níveis aceitáveis. As funções de gerenciamento devem compreender processos com características discretas, ou seja, com interações em pontos ou momentos determinados como, por exemplo o controle de admissão de uma tarefa. Também devem compreender processos com características dinâmicas, isto é, que atuem ao longo de todo o atendimento, possibilitando a identificação de alterações ocorridas no ambiente que possam prejudicar a QoS. Não é suficiente que a QoS desejada se limite a garantir apenas as especificações do cliente. De nada adiantam essas preocupações se a tecnologia utilizada (*hardware* e *software*) não oferece a confiabilidade e a segurança necessárias. Para tal, alguns parâmetros de qualidade baseados na tecnologia devem ser considerados, como por exemplo: latência, *jitter*, tempo de resposta, integridade, privacidade e disponibilidade.

Visto que as aplicações móveis são constituídas de programas que migram de um computador-cliente para um computador-servidor, com o propósito de serem executados no destino, políticas de QoS devem contemplar também os aspectos de mobilidade.

1.1.1 Trabalhos Relacionados

Os autores de [13] propõem uma política de controle de recursos para códigos móveis baseada no histórico. A idéia principal desta proposta é manter um histórico seletivo dos pedidos de acesso realizados e usar este histórico para aperfeiçoar o conhecimento entre pedidos seguros e pedidos potencialmente perigosos. Desta forma, o que será permitido ao programa vai depender da sua identidade, da sua origem e do seu comportamento durante a execução. A política de controle apresentada em [20] exemplifica o uso do domínio dos programas móveis como um dos atributos para liberação de recursos. A partir desse atributo e conforme as condições determinadas para cada domínio, podem ser impostas restrições do tipo “programas provenientes do servidor www.abcd.br devem consumir, no máximo, 30% de UCP” ou então “no mínimo 10% de tempo de UCP deve estar disponível aos programas procedentes de www.xyz.com.br”. A política de controle de recursos apresentada em [7] está fundamentada em uma aplicação de transação comercial entre agentes móveis. Essa proposta de controle de recursos baseia-se no modelo de moeda corrente. A moeda, que não precisa estar associada a uma moeda legal, é utilizada como elo de acordo entre os agentes e os serviços a serem prestados. Para isso, os agentes carregam uma quantidade finita de moeda que utilizam para pagar aos proprietários pela utilização dos recursos. Como resultado, os agentes têm a liberdade de escolher como e onde irão gastar sua moeda, e os proprietários que valor irão cobrar por cada serviço.

1.2 Controle de recursos para ambientes mistos

As políticas de gerenciamento de recursos da maioria dos sistemas operacionais convencionais não são capazes de controlar os requisitos de tempo. Por outro lado as políticas de escalonamento desenvolvidas para sistemas de tempo real não têm a mesma eficácia quando aplicadas em tarefas sem restrições de tempo. O desafio é encontrar uma política de escalonamento de recursos que satisfaça necessidades contraditórias: por um lado, é desejável que a

política tenha a flexibilidade e a imparcialidade (*fairness*) das políticas de escalonamento difundidas nos sistemas convencionais; por outro lado, deve oferecer a previsibilidade necessária às políticas de escalonamento de tarefas com restrições de tempo.

Os sistemas de computação, operando em ambientes dinâmicos, estão sujeitos a variações de carga e, portanto, sujeitos a condições de sobrecarga. Esta situação crítica, caracterizada pelo excesso de tarefas que demandam processamento, tem como maior risco o não cumprimento dos prazos previstos para as tarefas de tempo real. Se um sistema de tempo real, do tipo *hard-RT*, não está preparado para gerenciar sobrecargas, o efeito de um transiente deste tipo pode ser catastrófico. Em aplicações de tempo real onde as restrições de tempo não são rígidas (*soft-RT)*, como é o caso das aplicações multimídia, a perda do prazo traz como conseqüência, apenas, a degradação do desempenho. Essa degradação pode resultar, por exemplo, na perda de qualidade de uma apresentação de vídeo.

1.2.1 Trabalhos Relacionados

Até o momento não existe nenhuma taxonomia formal para as políticas de escalonamento destinadas aos ambientes mistos (os autores de [27] propõem uma taxonomia baseada em técnicas de escalonamento utilizadas). Todavia, dois modelos são muito difundidos nas políticas que buscam atender às necessidades desses ambientes. São elas: políticas de liberação de recursos baseadas em cotas proporcionais e políticas baseadas em reservas de recursos. As políticas baseadas em reserva de recursos oferecem um melhor suporte para o escalonamento de tarefas de tempo real, enquanto as baseadas em cotas proporcionais alcançam melhor flexibilidade e asseguram a distribuição justa entre as tarefas.

As políticas baseadas em reservas de recursos têm como premissa garantir a execução de uma tarefa através de algum mecanismo que aloque, com antecipação, o recurso computacional conforme a solicitação ou especificação da tarefa. As reservas de recursos podem ser realizadas com base no percentual de utilização [23] ou em unidades de tempo de UCP [17]. Os tempos livres, ou seja os tempos não reservados, são divididos entre as tarefas convencionais prontas para execução. Nas políticas baseadas em cotas proporcionais de [14, 31, 34], a liberação de recursos é proporcional ao *peso* que foi atribuído a cada tarefa. Se uma tarefa *A* tem o dobro do peso da tarefa *B*, então *A* vai receber duas vezes mais recursos do que a tarefa *B*. O valor da cota destinada a cada tarefa é então calculado com base no seu peso e na soma dos pesos de todas as tarefas ativas no momento.

De um modo geral, os algoritmos de escalonamento baseados em cotas proporcionais fundamentam-se na evolução do *tempo virtual* para decidir qual será a próxima tarefa a ser selecionada. O tempo virtual é um valor associado a cada tarefa de forma que sua taxa de crescimento é inversamente proporcional à sua cota e diretamente proporcional ao tempo de processamento consumido pela tarefa. Em linhas gerais, a tarefa que acumula o menor tempo virtual será a escolhida para execução. Considerando que a tarefa i iniciou sua execução no instante τ, o tempo virtual (v_i) da tarefa i no instante t é dado pela seguinte equação:

$$v_i(t) = v_i(\tau) + \frac{t - \tau}{s_i}$$

Onde:

$v_i(t)$: tempo virtual da tarefa i no instante t ;

$v_i(\tau)$: tempo virtual da tarefa i até o instante τ anterior;

s_i : cota da tarefa i durante o intervalo $[\tau, t)$

O projeto de um único esquema de escalonamento que atenda tanto a tarefas de tempo real quanto a tarefas convencionais tem provado ser de difícil realização. Com o objetivo de atender aos requisitos necessários aos ambientes mistos, algumas políticas híbridas têm sido propostas [12, 14, 20]. Nesses esquemas, as tarefas de tempo real são separadas das tarefas convencionais e, a cada um desses conjuntos, uma política diferente é aplicada. As políticas híbridas também se defrontam com as mesmas dificuldades encontradas em outros métodos quanto à dificuldade de propagar as decisões para um nível mais baixo do gerenciamento de recursos.

2 Uma Proposta de QoS para Programas Móveis

A proposta de QoS aqui apresentada representa uma política que permite diferenciar o nível do serviço a ser oferecido, conforme a classe do programa móvel a ser executado. Um exemplo da necessidade de se oferecer serviços diferenciados a estes programas é decorrente do crescimento de transações comerciais realizadas através da rede mundial de computadores, que normalmente envolvem acordos distintos entre o cliente e o servidor. Além disso, a elevada quantidade de programas móveis com características multimídia, ou seja, com requisitos de tempo, sustentam a necessidade de se tratar de forma adequada esses tipos de programas, além dos programas convencionais. Para esta política de QoS, o objetivo final é o gerenciamento de UCP, de acordo com os privilégios e as necessidades de cada programa.

2.1 Política de QoS

Para esta proposta foi considerada uma estrutura que apresenta a política de QoS em dois níveis (Figura 2.1): uma política que gerencia os requisitos de QoS no nível do cliente e um esquema de escalonamento de tarefas para ambientes mistos. Nesta estrutura, os parâmetros de QoS relacionados às restrições de processamento dos programas são avaliados e transportados para o esquema de escalonamento de tarefas. Em via contrária, as informações sobre a disponibilidade de recursos são fornecidas pelo esquema de escalonamento de tarefas.

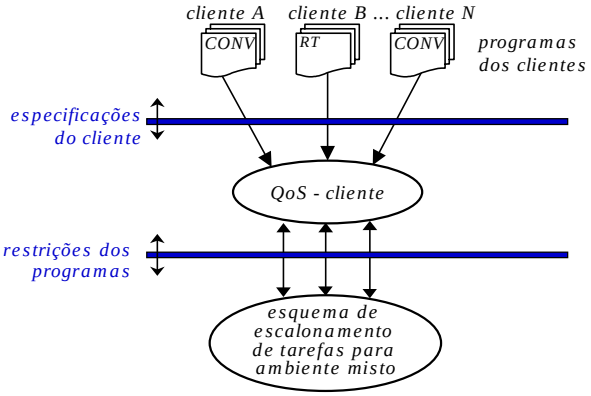


Figura 2.1 – Política de QoS em dois níveis.

2.1.1 Agrupamento hierárquico

Nesta política de QoS, os programas móveis são dispostos em uma estrutura hierárquica, em forma de árvore, que serve como base para a distribuição e contabilização do uso de recursos de UCP. A profundidade dessa estrutura hierárquica, ou seja, a quantidade de níveis dessa árvore, depende das especificações que se deva atender. Pode ser desejável uma estrutura com vários níveis onde, por exemplo, os programas pertencem a aplicações, que pertencem a departamentos e que, por sua vez, pertencem a empresas. Porém, qualquer que seja a estrutura criada, as *folhas* representam os programas móveis. Como em qualquer estrutura hierárquica, as restrições impostas aos níveis superiores são propagadas para os seus respectivos níveis inferiores. Na apresentação dessa proposta serão considerados apenas dois níveis hierárquicos: programas, ou tarefas, que pertencem a clientes. Cabe ressaltar que um programa ou tarefa pertence a um único cliente e que aqui o *cliente* é visto como uma abstração, podendo significar, por exemplo, uma aplicação, um domínio ou até mesmo uma empresa.

2.1.2 Política de distribuição de recursos

Com base nos conceitos de QoS mencionados no item 1.1, foram considerados os seguintes parâmetros baseados no cliente: (i) peso associado aos clientes; (ii) reserva de recursos para tarefas com restrições de tempo; (iii) cotas de recursos (proporcionais ao peso) para as tarefas convencionais; (iv) limites de uso do recurso; e (v) custo por unidade de tempo de UCP como forma de contabilização.

O *peso* atribuído ao cliente é utilizado como um instrumento que permite diferenciar o nível de serviço a ser oferecido. Portanto o valor atribuído ao peso deve fazer parte do acordo entre o cliente e o servidor. Em condições normais de carga, a liberação de recursos é proporcional ao peso de cada cliente. Dessa forma, se o cliente *A* tem o dobro do peso do cliente *B*, então *A* vai receber duas vezes mais recursos do que o cliente *B*. De modo similar, o mesmo mecanismo pode ser aplicado aos programas convencionais, ou seja: diferentes pesos podem ser atribuídos aos programas convencionais de um mesmo cliente, possibilitando o tratamento diferenciado também entre a execução de seus programas convencionais. Neste caso, cabe ao cliente a definição do peso de seus programas convencionais.

Para as tarefas com restrição de tempo, a garantia da execução dentro das especificações é alcançada através da reserva de recursos demandada pela própria tarefa, em conjunto com o algoritmo de escalonamento de tarefas. Esse algoritmo deve controlar tanto as reservas quanto as respectivas restrições de tempo das tarefas ativas. Para atingir tal objetivo, cada tarefa de tempo real deve especificar uma taxa de execução que assegure o resultado desejado. Desse modo, os pedidos de reserva devem ter a forma: *reserve Y unidades de tempo a cada X unidades para a tarefa i*. A partir dessa especificação, chega-se a uma cota fixa de utilização de UCP que estará disponível para a tarefa *i* até o fim da sua execução. Essa cota de UCP é fundamentada no cálculo do fator de utilização das tarefas periódicas [9, p.80], ou seja, a razão entre o tempo de computação e o período da tarefa ($U_i = C_i / P_i$).

Em sistemas *hard*-RT a garantia de execução da tarefa é baseada no tempo de execução de pior caso (*WCET*) da tarefa. Para esses casos, $C_i = WCET_i$. Porém, para aplicações *soft*-RT, onde podemos incluir as aplicações

multimídia, esse modelo pode causar um desperdício de recursos de UCP. Em aplicações do tipo "mídia contínua", como áudio e vídeo, o tempo para codificar e decodificar cada quadro pode variar significativamente. Dessa forma, o *WCET* da tarefa pode ser muito maior do que o tempo médio de execução da tarefa [1]. Como as restrições de tempo das tarefas *soft*-RT são mais relaxadas, a cota fixa referente ao pedido de reserva (r_i) de uma tarefa é baseada no tempo médio de execução da mesma. Considerando a sintaxe do pedido apresentada anteriormente: $r_i = Y_i / X_i$.

A estratégia de reservar uma cota fixa de utilização de UCP para as tarefas de tempo real resulta numa forma de isolamento entre tarefas. Ou seja, se uma determinada tarefa necessita de uma cota maior do que a cota reservada, apenas essa tarefa será executada mais lentamente, não prejudicando assim as demais tarefas. Os algoritmos baseados em reservas procuram assegurar que as tarefas não excedam às suas respectivas reservas.

O tempo de UCP não reservado, ou seja, o tempo restante, é dividido entre as tarefas convencionais de forma que a liberação de recursos seja proporcional ao *peso* atribuído a cada tarefa convencional. De uma forma geral, o valor da cota de recursos da tarefa *i* no instante *t* é dado pela divisão do peso da tarefa *i* pela soma dos pesos de todas as tarefas ativas no momento, ou seja, $s_i(t) = w_i / W$. Para este modelo hierárquico, a cota da tarefa convencional *i* deve levar em conta também o peso associado ao cliente. Para o cálculo da cota absoluta de uma tarefa convencional, teremos:

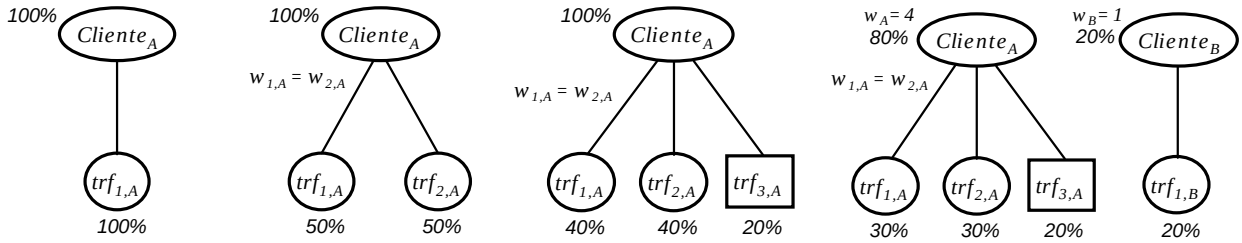
$$s_{i,j}(t) = \frac{w_{i,j}}{W_j} \text{ x } \left(\frac{h_j}{H} - R_j\right)$$

Onde:

$s_{i,j}(t)$	:	cota atribuída à tarefa convencional *i* do cliente *j* no instante *t* ;
$w_{i,j}$	:	peso associado à tarefa convencional *i* do cliente *j* ;
W_j	:	soma dos pesos das tarefas convencionais ativas do cliente *j* no instante *t* ;
h_j	:	peso associado ao cliente *j* ;
H	:	soma dos pesos de todos os clientes ativos no instante *t* ;
R_j	:	total de reservas do cliente *j* no instante *t* ;

Devido à dinâmica do ambiente em questão, caracterizada pela chegada e partida de tarefas ao longo do tempo, a cota de uma tarefa convencional pode variar ao longo de sua execução. Se considerarmos o mesmo *peso* para as tarefas convencionais de um determinado cliente, as cotas de UCP das suas tarefas convencionais ativas irão diminuir à medida que a quantidade de tarefas ativas cresça, ou seja, à medida que cheguem novas tarefas convencionais. Para esta consideração, a recíproca é verdadeira.

A integração da reserva de recursos das tarefas de tempo real com as cotas proporcionais das tarefas convencionais considerada nesta proposta, foi motivada por um fundamento simples e evidente, apresentado em [32]. Em linhas gerais, a estrutura apresentada sempre mantém as cotas correspondentes às reservas das tarefas de tempo real, distribuindo o restante, ou seja, o tempo não reservado, de modo proporcional ao peso de cada tarefa convencional ativa. Em outras palavras, mesmo em um ambiente dinâmico, as cotas das tarefas de tempo real permanecem fixas, enquanto que as cotas das tarefas convencionais variam de modo a se ajustarem ao tempo não reservado.



(a) passo 1 (b) passo 2 (c) passo 3 (d) passo 4

Figura 2.2 – Exemplo da integração de cotas proporcionais com reserva de recursos. As cotas percentuais indicadas nas figuras representam valores absolutos.

Como exemplo, considere primeiramente apenas um cliente executando uma única tarefa convencional ($trf_{1,A}$). Neste momento, tanto o cliente quanto a sua tarefa têm disponíveis 100% de tempo de UCP (Figura 2.2.a). Em um segundo passo, uma nova tarefa convencional ($trf_{2,A}$), pertencente ao mesmo cliente, se torna ativa. Considerando que as duas tarefas têm o mesmo peso ($w_{1,A} = w_{2,A}$) e que disputam o mesmo recurso, caberá a cada uma das tarefas

50% de tempo de UCP (Figura 2.2.b). Num próximo passo, uma tarefa com restrição de tempo ($trf_{3,A}$), ainda do mesmo cliente, se torna ativa. Para essa última, é imperativo a reserva de 20% de tempo de UCP uma vez que a sua taxa de execução foi especificada em 40*ms* a cada 200*ms*, por exemplo. A partir desse momento, caberá a cada uma das duas tarefas convencionais ($trf_{1,A}$ e $trf_{2,A}$) 40% de tempo de UCP, já que 20% estão reservados para a tarefa de tempo real (Figura 2.2.c). Se em um próximo passo uma nova tarefa se torna ativa ($trf_{1,B}$), independentemente de ser uma tarefa convencional ou não, ou até mesmo de pertencer a um outro cliente, a reserva de 20% da tarefa de tempo real $trf_{3,A}$ será mantida (Figura 2.2.d).

2.1.3 Controle de admissão

Para que haja garantia de execução das tarefas de tempo real, o controle de admissão, ou teste de aceitabilidade, se faz necessário, uma vez que as reservas solicitadas para estas tarefas não devem exceder a 100% do recurso. Como o ambiente também se compromete a atender tarefas convencionais, é interessante estabelecer um limite percentual (não necessariamente fixo) relacionado ao total de reservas das tarefas de tempo real, de modo que sempre haja recurso para as tarefas convencionais. De um modo geral, o teste de admissão para as tarefas com restrição de tempo, ou seja, para as tarefas *soft*-RT, deve atender à seguinte equação:

$$\sum_{i=1}^{n} r_i \leq limite_{RT}$$

Onde:

r_i : fração da reserva da tarefa *i* (item 2.1.2);

$limite_{RT}$: valor entre [0...1] definido como o limite máximo de uso de UCP para as tarefas com restrição de tempo.

Este limite pode ser definido, por exemplo, com base em estatísticas de uso de recursos do próprio servidor. Ou seja, se a demanda de um determinado servidor é por tarefas convencionais, reduz-se o limite de reservas das tarefas de tempo real. Além disso, a política de uso também pode contemplar limites específicos para cada cliente, onde os limites podem ser definidos pelos próprios clientes ou de comum acordo com o servidor.

2.1.4 Negociação da reserva

Para esta proposta, a negociação tem como objetivo viabilizar a execução de uma tarefa com restrição de tempo que *a priori* foi rejeitada pelo teste de admissão. Se as condições atuais não permitem atender a taxa de execução da tarefa rejeitada, talvez seja possível atendê-la caso a mesma relaxe o rigor das suas restrições de tempo. Para agilizar a negociação, a política de QoS deve dispor de procedimentos que permitam conhecer a atual disponibilidade de recursos, entre outras informações de interesse. As indagações sobre a disponibilidade de recursos normalmente são feitas ao escalonador de tarefas e podem ser implementadas através de funções do tipo:

`int limite_reservas()` : retorna o valor atual atribuído ao limite de reservas das tarefas de tempo real,

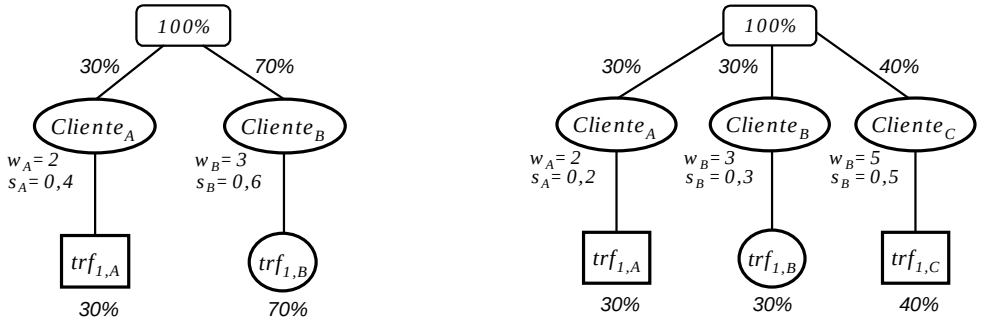
`int total_reservado()` : retorna o valor total atual de reservas.

2.1.5 Política de custos para injustiças momentâneas

A integração de reserva de recursos com cotas proporcionais, em um ambiente altamente dinâmico, pode resultar em injustiças momentâneas com relação à proporcionalidade do peso do cliente e à sua respectiva cota de recursos. Considere as seguintes situações:

Situação 1: duas tarefas de diferentes clientes estão ativas e disputando o mesmo recurso (Figura 2.3.a). A tarefa de tempo real do cliente *A* ($trf_{1,A}$) solicitou uma reserva de 30% de tempo de UCP, significando uma cota fixa de 30%. O cliente *B* está executando uma tarefa convencional ($trf_{1,B}$) que está tomando os 70% restantes. Se considerarmos que os pesos (w) dos clientes *A* e *B* são respectivamente 2 e 3 podemos concluir que o cliente *A* está abaixo da cota que lhe é permitida, uma vez que cabe a ele neste momento 40% ($s_A = 2/5$) de tempo de UCP. Conseqüentemente o cliente *B* ($s_B = 3/5$) está usufruindo de uma cota maior do que lhe cabe neste momento.

Situação 2: a partir da situação anterior, uma nova tarefa de tempo real se torna ativa ($trf_{1,C}$). A taxa de execução dessa tarefa corresponde a uma reserva de 40% de UCP. Considerando que essa tarefa pertence ao cliente *C* de peso *5*, observamos que o mesmo cliente *A*, que na situação anterior estava sub-utilizando sua cota proporcional, passou a excedê-la em 10%. Nesta situação, cabe ao cliente *A* apenas 20% de UCP ($s_A = 2/10$). A Figura 2.3.b ilustra esta situação.



(a) situação 1 (b) situação 2

Figura 2.3 – Exemplo de situações com injustiças momentâneas. As cotas percentuais indicadas nas figuras representam valores absolutos.

Essas injustiças momentâneas podem ser contornadas implementando-se uma política de custo baseada nas condições de carga do ambiente. Considerando o exemplo descrito anteriormente, não seria correto retirar os 10% excedentes do cliente *A* (na situação 2) uma vez que a sua tarefa com restrições de tempo foi aceita e portanto lhe foi dada a garantia de execução. A solução proposta é aumentar o custo por unidade de tempo de UCP quando o cliente exceder sua cota. Com isso, o cliente pode optar em ter um custo mais elevado ou aguardar para realizar o processamento em períodos que o sistema esteja com menor demanda de utilização. A médio ou longo prazo, a incidência desse fato (o cliente exceder sua cota) poderá levar a uma renegociação no valor do $peso$ atribuído ao cliente ou, até mesmo, a uma reavaliação da disponibilidade do servidor.

2.2 Algoritmo de Escalonamento de Tarefas para Ambiente Misto

O algoritmo de escalonamento aqui apresentado se aplica a um ambiente multitarefas, uniprocessado e preemptivo. Seu objetivo é manter a taxa de execução das tarefas de tempo real, do tipo *soft*-RT, e também atender às tarefas convencionais, independentemente da quantidade de tarefas ativas e da dinâmica do ambiente. Manter a taxa de execução de uma tarefa de tempo real significa executar Y unidades de tempo a cada X unidades, durante toda a existência dessa tarefa. Como as restrições de tempo das tarefas *soft*-RT não são rígidas, a perda eventual do seu prazo de execução é plenamente tolerável. Ou seja, executar $Y \pm \Delta Y$ unidades de tempo por alguns períodos de X unidades, é tolerável.

O algoritmo considera uma quantidade fixa de tempo, denominada *quantum* (Q), dedicada à execução de uma tarefa sem que ocorra interrupção. Desta forma, no início de cada *quantum* uma tarefa é selecionada para usar o tempo de UCP e, uma vez que a tarefa tenha adquirido o recurso, ela poderá usá-lo por até esse *quantum*. Em suma, o algoritmo de escalonamento emula o sistema de escoamento de fluido em um sistema de alocação discreta de tempo [26], onde o recurso é distribuído em unidades discretas de tempo de comprimento Q.

O valor do *quantum*, definido pelo esquema de escalonamento, deve ser suficientemente pequeno a fim de garantir a granularidade desejável ao atendimento das tarefas de tempo real, porém não tão pequeno que resulte em um chaveamento de contexto excessivo. Convém mencionar que o término da execução de uma tarefa antes do término do seu *quantum* apenas adianta o início do processamento da próxima tarefa.

Como definido no item 2.1.2, sobre a política de distribuição de recursos, cotas fixas de UCP são associadas às tarefas com restrição de tempo e cotas variáveis são associadas às tarefas convencionais. Desse modo, o algoritmo de escalonamento aqui proposto utiliza as cotas fixas e variáveis dessas tarefas como base para obtenção da escala de execução de tarefas. Para tal, a cota de cada tarefa é utilizada para efetuar a evolução de dois tempos virtuais: um tempo virtual relacionado ao tempo consumido na execução da tarefa, denominado *TVC*, e um tempo virtual relacionado ao prazo final das tarefas de tempo real, denominado *TVP*. Neste algoritmo, o escalonador destina o novo *quantum* para a tarefa que acumula o menor tempo virtual de referência (*TVR*), tempo esse obtido através da adição dos dois outros tempos virtuais: o *TVC* e o *TVP*.

O conceito de $cotas$, $pesos$ e $tempo\ virtual$, como base de políticas de escalonamento, não é uma inovação. Como apresentado no item 1.2.1, este conceito é assunto de vários artigos. Porém, modificações devem ser efetuadas nas formas tradicionais visando contemplar as restrições de tempo das tarefas de tempo real.

O algoritmo de escalonamento proposto tem como característica a facilidade de implementação em sistemas operacionais de tempo compartilhado, além do emprego de uma única lógica de controle de recursos, aplicada tanto sobre as tarefas convencionais, quanto sobre as tarefas de tempo real.

2.2.1 Composição das filas de tarefas: principal e secundária

Neste esquema de escalonamento, as tarefas são dispostas em duas filas: uma fila principal e uma fila secundária. Na fila principal ($Fila_{PRI}$) são colocadas as tarefas de tempo real ativas (*s-RT*), além de uma tarefa ociosa, aqui chamada de *REST*, relacionada ao controle do tempo restante. A fila secundária ($Fila_{SEC}$), é destinada às tarefas convencionais ativas (*CONV*). Como o tempo restante é destinado às tarefas convencionais, a tarefa ociosa *REST*, deve apontar para a fila secundária. A Figura 2.4 ilustra este relacionamento.

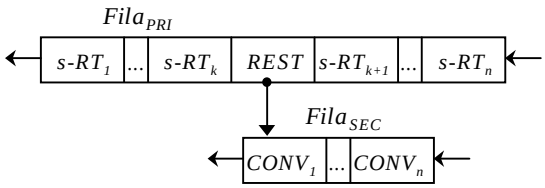


Figura 2.4 – Relacionamento entre as filas de tarefas principal ($Fila_{PRI}$) e a secundária ($Fila_{SEC}$).

Ambas as filas são ordenadas pelo tempo virtual de referência (*TVR*). A tarefa ativa que possui o menor *TVR*, ocupa a cabeça da fila e portanto será a próxima tarefa a ser selecionada para execução. O algoritmo opera primeiramente sobre a fila principal, de modo que: se a tarefa que ocupa a cabeça desta fila é uma tarefa de tempo real, e não a tarefa ociosa *REST*, o próximo *quantum* é destinado à respectiva tarefa de tempo real. Caso contrário, o próximo *quantum* é destinado à tarefa que ocupa a cabeça da fila secundária, ou seja, destinado a uma tarefa convencional.

2.2.2 Evolução dos tempos virtuais TVC e TVP

Os tempos virtuais *TVC* e *TVP* e suas respectivas evoluções asseguram a seqüência de execução das tarefas conforme as características especificadas pelas tarefas de tempo real e pelas tarefas convencionais. A taxa de acréscimo utilizada na evolução dos tempos virtuais, tanto do *TVC* quanto do *TVP* de cada tarefa, é inversamente proporcional à cota associada à tarefa ($1/s_i$). Entretanto, suas evoluções ocorrem de formas distintas. Como o *TVC* está associado ao tempo de computação da tarefa, o seu valor só é incrementado quando a tarefa é selecionada para execução. Dessa forma, o valor do *TVC* de uma tarefa permanece constante enquanto a mesma estiver na fila de tarefas ativas, ou seja, aguardando ser chamada pelo escalonador.

O *TVC* também é associado às duas filas de tarefas: $Fila_{PRI}$ e $Fila_{SEC}$. O *TVC* da fila é incrementado sempre que uma de suas tarefas é selecionada para execução. A cada novo *quantum*, uma das tarefas da fila principal é selecionada para execução (ou a tarefa *REST*, ou uma tarefa de tempo real). Dessa forma, o *TVC* da $Fila_{PRI}$ será incrementado sempre no início de cada *quantum*. Diferentemente, o *TVC* da $Fila_{SEC}$ só será incrementado quando uma das tarefas da sua fila é selecionada para execução. Sob outro ponto de vista, o *TVC* da $Fila_{SEC}$ é incrementado sempre que a tarefa *REST*, da fila principal, é selecionada para execução.

O valor da taxa de acréscimo do *TVC* de cada fila é inversamente proporcional ao somatório das cotas das tarefas da própria fila. Conseqüentemente, a taxa de acréscimo do *TVC* da $Fila_{PRI}$, é sempre de uma unidade, dado que o somatório das cotas das tarefas dessa fila é sempre igual a unidade. Para a $Fila_{SEC}$, a taxa de acréscimo do seu *TVC* é inversamente proporcional a cota da tarefa *REST*, visto que essa tarefa representa o total das cotas das tarefas da fila secundária. A necessidade de se associar o *TVC* também a estas filas ($Fila_{PRI}$ e $Fila_{SEC}$) está relacionada com a chegada de tarefas em diferentes momentos, ou seja, com a dinâmica do ambiente. Para isso, o *TVC* da tarefa recebe o valor do *TVC* da sua respectiva fila quando a tarefa se torna ativa, ou seja, quando é inserida na fila ($Fila_{PRI}$ ou $Fila_{SEC}$). Este procedimento assegura que as tarefas que estão em atividade há algum tempo não tenham sua execução prejudicada pela tarefa que acabou de chegar.

Quanto ao *TVP*, o seu valor (em unidades de *quantum)* revela o término do prazo para execução de uma unidade de tempo da tarefa. Dessa forma, a cada novo *quantum*, o *TVP* de cada tarefa deve sempre ser decrementado de uma unidade. Em contrapartida, o *TVP* de uma tarefa é incrementado da sua taxa de acréscimo somente quando a tarefa é selecionada para execução, significando, assim, um novo prazo para a execução de mais uma unidade de tempo da respectiva tarefa. Como o *TVP* está relacionado ao término do prazo de uma tarefa, sua evolução se aplica somente às tarefas de tempo real. Conseqüentemente, o *TVP* existe para todas as tarefas da fila principal, inclusive para a tarefa ociosa *REST*. Convém lembrar que a taxa de incremento do *TVP* é calculada com base na cota associada à tarefa, ou seja, $1/s_i$. Dessa forma, o *TVP* da tarefa *REST*, mesmo não sendo uma tarefa de tempo real, pode ser plenamente evoluído, uma vez que essa tarefa representa a cota restante. Para a fila secundária, o valor do *TVP* das tarefas é igual a zero. Logo, o valor do tempo virtual de referência (*TVR*) de cada tarefa convencional é igual ao valor do próprio *TVC*, uma vez que *TVR = TVP + TVC*.

3 Implementação do Algoritmo no Ambiente Java-Linux

Para a implementação do algoritmo de escalonamento proposto, foi considerado o modelo *green-threads* da versão 1.2.2 para Linux, da Sun Microsystems, Inc [30]. Para tal versão, as características relevantes a essa implementação são:

- as prioridades associadas às Java-threads, definidas pelo programador, são transportadas para o escalonador da JVM sem qualquer tipo de adaptação. A escala de prioridades dessas threads varia de MIN_PRIORITY até MAX_PRIORITY.
- existem dois tipos de *threads* na JVM: as *system-threads* e as *user-threads*. As *system-threads* são criadas quando a JVM é requerida. Essas *threads* realizam tarefas relacionadas à operação da JVM, como por exemplo, o gerenciamento de tempo (*clock*). As *user-threads* são criadas para executar os programas do usuário.
- as prioridades das *system-threads* são baseadas nos valores definidos em MIN_PRIORITY e MAX_PRIORITY. Por exemplo, a *thread* responsável pelo gerenciamento de tempo, *clockHandler( )*, roda com uma prioridade igual a MAX_PRIORITY + 2. A *clockHandler( )* é a *thread* de maior prioridade dentre todas as *threads* da JVM.
- todas as *threads* (*user-threads* e *system-threads*) que se encontram no estado RUNNABLE são dispostas numa fila, *runnable_queue*, ordenada por prioridade. O escalonador da JVM, *rescheduler( )*, busca a *thread* de maior prioridade.
- o compartilhamento de tempo de UCP entre *threads* de mesma prioridade somente é obtido com a ativação da função *timeSlicer( )*, iniciada com a passagem do parâmetro *-Xts* para o interpretador Java.
- A função *timeSlicer( )*, de prioridade MAX_PRIORITY + 1, trabalha em conjunto com a função *clockHandler( )*, controlando o tempo de compartilhamento entre *threads* de mesma prioridade.

O esquema de escalonamento aqui proposto não considera o atributo *prioridade* como base da distribuição de recursos dos programas dos usuários, ou seja, das *user-threads*. No entanto, existe a necessidade de se priorizar a execução de determinadas *threads*, as *system-threads*, em favor das *user-threads*, e vice-versa. Por exemplo: a *thread* responsável pelo gerenciamento do tempo, a *clockHandler( )*, deve sempre ser executada antes de qualquer *user-thread*. De forma contrária, a *thread* ociosa da JVM, a *idle_loop( )*, só deve ser executada quando nenhuma *user-thread* estiver na fila *runnable_queue*. Assim sendo, a implementação do algoritmo manteve o atributo *prioridade* associado às *threads*, porém, todas as *user-threads* possuem a mesma prioridade. Essa determinação não afeta o modelo de escalonamento da JVM original, mas transfere o controle da distribuição de recursos das *user-threads* para outros parâmetros, como é o nosso objetivo.

De uma forma geral, as modificações necessárias à implementação do algoritmo proposto foram realizadas nos seguintes pontos: (i) no descritor das *threads*, (ii) na forma de organização da fila *runnable_queue* e (iii) na função responsável pela seleção da próxima *thread* para execução, a função *rescheduler( )*.

- Descritor da *thread*: o descritor foi modificado a fim de incorporar informações necessárias ao funcionamento do novo esquema de escalonamento. Para tal, foram adicionados os campos `cota`, `tvc`, `tvp`, `tvr`, dentre outros.
- Fila *runnable_queue*: a forma de organização da fila *runnable_queue* passa a considerar o campo `tvr`, além do campo `priority` existente no descritor da *thread* da JVM original. Na nova implementação, as *threads* dessa fila são dispostas em ordem decrescente de prioridade (campo `priority`), seguidas pela ordem crescente do tempo virtual de referência (campo `tvr`).
- Função *rescheduler*(): as modificações realizadas nessa função tiveram como objetivo principal atualizar os cálculos dos valores virtuais *TVC, TVP* e *TVR* das *user-threads*.

Também foi implementada uma política de controle para as *user-threads* que se encontravam no estado SUSPENDED, antes destas serem incluídas na fila *runnable_queue*. Uma *thread* está no estado SUSPENDED quando está aguardando por um determinado evento. Este controle visa assegurar que as *user-threads* em atividade não sejam prejudicadas por um possível valor baixo de *TVC* das *user-threads* que se encontravam no estado SUSPENDED.

4 Resultados dos Experimentos

De forma a avaliar o desempenho do algoritmo de escalonamento, vários testes foram realizados em uma estação de trabalho com um processador da AMD, modelo Athlon de 750MHz, com 128K bytes de memória RAM. Para esses testes foram criadas tarefas (Java-*threads*) do tipo UCP-intensiva, na forma de uma repetição de comandos ("*loop* de comandos"). O número total de repetições e o tempo atual foram utilizados como métrica de desempenho da

implementação. Foi introduzida nessa repetição uma variável aleatória uniformemente distribuída em um intervalo para que o tempo gasto por essa função não fosse constante. O objetivo da inclusão dessa variável foi tornar o programa de teste mais próximo do que ocorre no processamento de imagens, onde o tempo de processamento de cada quadro pode variar significativamente. A representação gráfica da variação do tempo de processamento de cada repetição pode ser visto na Figura 4.1.

Nesse trabalho, por uma questão de espaço, indicamos os resultados de parte dos experimentos realizados. Para o teste aqui indicado, foi monitorado o número de repetições de quatro tarefas (Java-*threads*) disparadas em momentos distintos. Dessas quatro, três tarefas possuem características de tempo real. Assim sendo, necessitam reservar recursos de UCP para o atendimento de suas especificações. As tarefas de tempo real, *RT-A*, *RT-B* e *RT-C*, demandaram, respectivamente, as seguintes taxas de execução: 50*ms* a cada 500*ms* (o equivalente a $2fps$, dois *frames* por segundo), 50*ms* a cada 200*ms* ($5fps$) e 50*ms* a cada 100*ms* ($10fps$). Essas taxas resultaram na reserva de 10% para a tarefa *RT-A*, 25% para a *RT-B* e 50% para a *RT-C*. O tempo de compartilhamento de UCP, ou seja, o valor associado ao parâmetro $-Xts$, foi de 10*ms*. A primeira tarefa disparada foi a tarefa convencional *CV-X*. As demais tarefas, *RT-A*, *RT-B* e *RT-C*, foram disparadas, nesta ordem, com intervalos de 10 segundos em relação à tarefa anteriormente disparada. Com base no resultado do teste (Figura 4.2), podemos observar que cada tarefa de tempo real manteve sua respectiva taxa de execução durante toda a sua atividade (representada pela mesma inclinação durante todo o seu tempo de execução). Como esperado, a única tarefa que teve a sua taxa de execução alterada foi a tarefa convencional *CV-X*, resultado das necessárias adaptações às mudanças de carga do sistema.

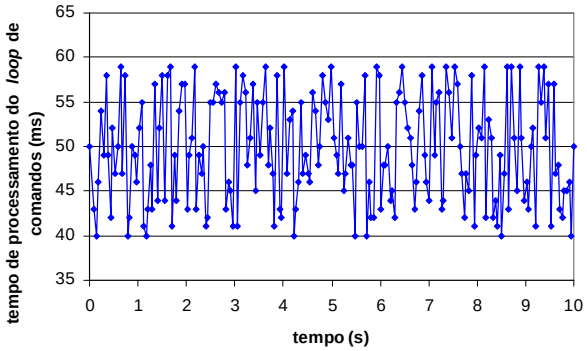


Figura 4.1 – Variação do tempo de processamento consumido pelo "*loop* de comandos".

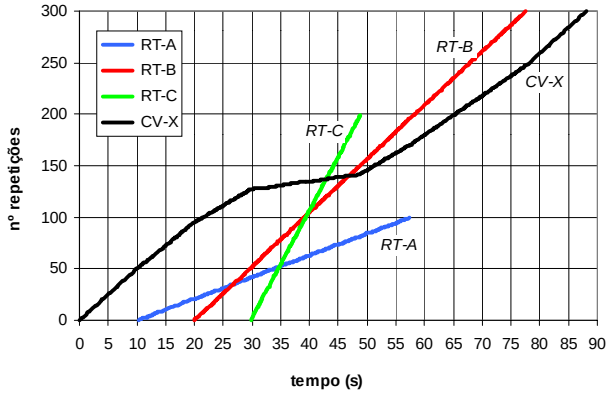


Figura 4.2 – Número de repetições de quatro tarefas, sendo: três de tempo real (*RT-A*, *RT-B* e *RT-C*) e uma convencional (*CV-X*).

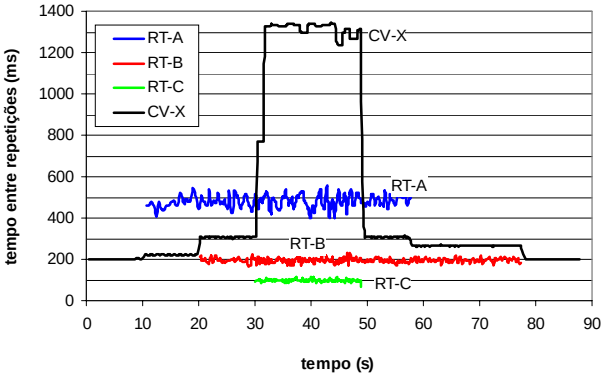


Figura 4.3 – Tempo entre repetições (período) das tarefas de tempo real *RT-A*, *RT-B* e *RT-C* e da tarefa convencional *CV-X*.

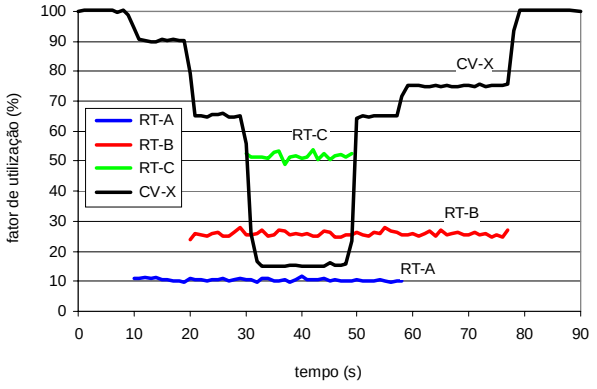


Figura 4.4 – Fator de utilização de UCP das tarefas de tempo real *RT-A*, *RT-B* e *RT-C* e da tarefa convencional *CV-X*.

Através da Figura 4.3 também podemos verificar todo o processo de adaptação da tarefa convencional. Nessa figura, podemos constatar que o intervalo entre 30 e $50ms$, aproximadamente, foi o mais lento de sua execução, quando resultou numa média entre repetições de $1300ms$. Finalizando os resultados para este teste, a Figura 4.4 apresenta o fator de utilização da UCP de cada tarefa, ao longo do tempo.

5 Conclusão

Neste trabalho foi apresentada uma proposta de política de QoS para programas móveis, tendo como resultado final a liberação de UCP de acordo com as características e necessidades de cada programa móvel. Como parte dessa política, foi apresentado e implementado um algoritmo de escalonamento de tarefas orientado ao atendimento de tarefas de tempo real (do tipo soft-RT) em conjunto com tarefas convencionais (sem requisitos de tempo real). Em outras palavras, um algoritmo de escalonamento para ambiente misto. Sendo assim, a política de QoS considerou a procedência e as restrições de execução de cada programa móvel, levando em conta a característica dinâmica dos ambientes de programas móveis e mantendo a flexibilidade das políticas encontradas nos sistemas operacionais de uso geral. Neste sentido, a política de QoS proposta compreendeu:

- **um agrupamento hierárquico de clientes:** serve de base para a distribuição e contabilização do uso de recursos de UCP;
- **a atribuição de *peso* aos clientes:** utilizado como um instrumento de diferenciação dos níveis de serviço a ser oferecido;
- **a reserva de recursos para tarefas com restrições de tempo:** a manutenção da taxa de execução da tarefa de tempo real é garantida através da reserva de recursos demandada pela própria tarefa;
- **diferentes cotas de recursos para as tarefas convencionais de um mesmo cliente**: possibilita o tratamento diferenciado entre os programas convencionais de um mesmo cliente;
- **um controle de admissão:** se faz necessário, uma vez que as reservas solicitadas para estas tarefas não devem exceder a 100% do recurso;
- **limites de uso do recurso:** considera um limite percentual relacionado ao total de reservas das tarefas de tempo real, de modo que sempre haja recurso para as tarefas convencionais;
- **a negociação da reserva:** viabiliza a execução de uma tarefa com restrição de tempo que *a priori* foi rejeitada pelo teste de admissão;
- **uma política de custo no caso de sobrecarga:** solução dada às "injustiças momentâneas"; e
- **um algoritmo de escalonamento para ambiente misto:** visa manter a taxa de execução das tarefas de tempo real além de atender às tarefas convencionais, independentemente da variação de carga do ambiente. O algoritmo proposto é simples e de fácil implementação.

Finalmente, alguns experimentos para avaliação do algoritmo de escalonamento aqui proposto foram apresentados. Os testes realizados nesses experimentos foram encorajadores. Assim, como desenvolvimento da proposta aqui apresentada estamos planejando a realização de avaliações utilizando tarefas típicas de aplicações multimídia reais.

Referências

[1] Abeni, L. e Buttazzo, G. Integrating Multimedia Applications in Hard Real-Time Systems. *IEEE Real-Time Systems Symposium*, Madri, Espanha, (Dezembro, 1998).

[2] Abeni, L. e Buttazzo, G. Adaptive Bandwidth Reservation for Multimedia Computing. *IEEE Real-Time Computing Systems and Applications Conference*, Hong Kong, China, (Dezembro, 1999).

[3] Abeni, L. e Buttazzo, G. Hierarchical QoS Management for Time Sensitive Applications. *IEEE Real-Time Technology and Applications Symposium*, Taipei, Taiwan, (Maio, 2001).

[4] Abeni, L. e Buttazzo, G. Constant Bandwidth vs Proportional Share Resource Allocation. *IEEE International Conference on Multimedia Computing and Systems*, Florença, Itália, (Junho, 1999).

[5] Aurrecoechea, C., Campbell, A. e Hauw, L. A Survey of QoS Architectures. *ACM Multimedia Systems Journal*, (Maio, 1998).

[6] Bennett, J.C.R., Stephens, D.C. e Zhang, H. High Speed, Scalable, and Accurate Implementation of Packet Fair Queueing Algorithms in ATM Networks, *IEEE INFOCOM '96*, São Francisco, EUA, (1997).

[7] Bredin, J., Kotz, D. e Rus, D. Market-based Resource Control for Mobile Agents. *Proceedings of the Second International Conference on Autonomous Agents*, Minneapolis, EUA, (Maio, 1998).

[8] Burns, A. e Wellings, A. *Real-Time Systems and Programming Languages*. 2ª ed., Addison-Wesley. 1996.

[9] Buttazo, G. *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications.* Kluwer Academic Publishers. 1997.

[10] Chalmers, D. e Sloman, M. A Survey of Quality of Service in Mobile Computing Environments. *IEEE Communications Surveys*, Vol. 2, Nº 2, (1999).

[11] Cisco Systems. Quality of Service (QoS) Networking. *Internetworking Technology Overview*, (Junho, 1999).

[12] Deng, Z. e Liu, J.W.S. Scheduling Real-Time Applications in an Open Environment. *IEEE Real-Time Systems Symposium*, (Dezembro, 1997).

[13] Edjlali, G., Acharya, A. e Chaudhary, V. History-based Access Control for Mobile Code. *Fifth ACM Conference on Computer and Communications Security*, São Francisco, EUA, (Novembro, 1998).

[14] Goyal, P., Guo, X. e Vin, H.M. A Hierarchical CPU Scheduler for Multimedia Operating Systems. *USENIX 2nd Symposium on Operating Systems Design and Implementation*, (Outubro, 1996).

[15] Henry, G.J. The Fair Share Scheduler. *AT&T Bell Laboratories Technical Journal*. Vol. 63, Nº 8, (Outubro, 1984), p.p.1845-1858.

[16] Jones, M.B. *et al*. An Overview of Rialto Real-Time Architecture. *Seventh ACM Symposium on Operating Systems Principles*, Irlanda, (Setembro, 1996), p.p. 249-256.

[17] Jones, M.B., Rosu, D. e Rosu, M.C. CPU Reservations and Time Constraints: Efficient, Predictable Scheduling of Independent Activities. *16th ACM Symposium on Operating Systems Principles*, Saint-Malo, França, (Outubro, 1997), p.p. 198-211.

[18] Kay, J. e Lauder, P. A Fair Share Scheduler. *Communications of the ACM*, Vol. 31, Nº1, (Janeiro, 1988), p.p. 44-55.

[19] Kopetz, H. *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Kluwer Academic Publishers, (1997).

[20] Lal, M. e Pandey, R. CPU Resource Control for Mobile Programs. *First International Symposium on Agent Systems and Applications*, (Outubro, 1999).

[21] Lipari, G. e Abeni, L. Sharing Resources in an Open System. *Technical Report RETIS TR2001-05*, (Setembro, 2001).

[22] Liu, C.L. e Layland, J.W. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. *Journal of the Association for Computing Machinery*, Vol. 20, Nº 1, (1973).

[23] Mercer, C.W., Savage, S. e Tokuda, H. Processor Capacity Reserves: Operating System Support for Multimedia Applications. *IEEE International Conference on Multimedia Computing and Systems*, Boston, EUA, (Maio, 1994), p.p. 90-99.

[24] Nieh, J. e Lam, M.S. The Design, Implementation and Evaluation of SMART: A Scheduler for Multimedia Applications. *16th ACM Symposium on Operating Systems Principles*, Saint-Malo, França, (Outubro, 1997).

[25] Oaks, S. e Wong, H. *Java Threads*. 2ª ed., O'Reilly, (Janeiro, 1999).

[26] Parekh, A.K. e Gallager, R.G. A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Single-Node Case, *IEEE/ACM Transactions on Networking*, (Junho, 1993).

[27] Regehr, J., Jones, M.B e Stankovic, J. Operating System Support for Multimedia: The Programming Model Matters. *Microsoft Research Technical Report MSR-TR-2000-89*, (Setembro, 2000).

[28] Stankovic, J. Misconceptions About Real-Time Computing. *IEEE Computer*, Vol.21, Nº10, (Outubro, 1988).

[29] Stankovic, J., Spuri, M., Di Natale, M. e Buttazzo, G. *Implication of Classical Scheduling Results for Real-Time Systems*. (Junho, 1994).

[30] Sun Microsystems, Inc. <www.sun.com> <java.sun.com>.

[31] Stoica, I. *et al*. A Proportional Share Resource Allocation Algorithm for Real-Time, Time-Shared Systems. *IEEE 17th Real-Time Systems Symposium*, Washington, EUA, (Dezembro, 1996), p.p. 288-299.

[32] Stoica, I., Abdel-Wahad, H. e Jeffay, K. On the Duality between Resource Reservation and Proportional Share Resource Allocation. *Multimedia Computing and Networking*, Vol. 30, San Jose, EUA, (Fevereiro, 1997), p.p. 207-214.

[33] Waldspurger, C.A. e Weil, W.E. Lottery Scheduling: Flexible Proportional-Share Resource Management. *1st OSDI Symposium*, (Novembro, 1994), p.p. 1-12.

[34] Waldspurger, C.A. e Weil, W.E. Stride Scheduling: Deterministic Proportional Share Resource Management. *Technical Memorandum, MIT/LCS/TM-528*, Laboratory for CS, MIT, (Julho, 1995).