QoSOSLinux: Desenvolvimento de uma extensão Linux com suporte adequado a qualidade de serviço

Marcelo Ferreira Moreno[1], Luiz Fernando Gomes Soares[1]

[1]Dep. de Informática – Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio)
Rua Marquês de São Vicente, 225 RDC – 22453-900 – Rio de Janeiro – RJ – Brasil
{moreno,lfgs}@inf.puc-rio.br

Abstract. This paper presents the main design issues for extending a general-purpose operating system (GPOS) towards an appropriate support for distributed multimedia applications. The goal is achieved by resource management mechanisms based on quality of service (QoS) guarantees. The QoSOS architecture, developed in previous work, is the basis for the implementation of a new version of the Linux operating system, called QoSOSLinux.

Resumo. Este artigo apresenta os principais aspectos de projeto para a extensão de um sistema operacional de propósito geral (GPOS), rumo ao suporte adequado a aplicações multimídia distribuídas, através da gerência de recursos com garantias de qualidade de serviço (QoS). A arquitetura QoSOS, resultado de trabalhos anteriores, foi utilizada como base para a implementação de uma versão do sistema operacional Linux, denominada QoSOSLinux.

1. Introdução

A notável evolução das técnicas de codificação de mídias contínuas vem tornando cada vez mais difundido o uso de aplicações multimídia distribuídas. Nessas aplicações, para que o usuário tenha uma interação satisfatória com o sistema, se faz necessário o atendimento a requisitos como retardo máximo, vazão e jitter, de maneira fim-a-fim. Deve-se garantir, portanto, que por todo o caminho de comunicação haverá recursos disponíveis para encaminhamento e processamento adequados dos dados. Essa funcionalidade é alcançada por meio de mecanismos de gerência de qualidade de serviço (QoS – quality of service) em todos os subsistemas participantes.

Presentes em todos os nós da rede (estações finais, roteadores ou comutadores), os sistemas operacionais representam pontos centrais de gerenciamento de recursos de processamento, armazenamento e comunicação de dados, e, portanto, têm um papel fundamental na provisão de QoS. Nesse cenário fim-a-fim, pode-se destacar ainda a popularidade dos sistemas operacionais de propósito geral (GPOS – *general purpose operating systems*), presentes principalmente nas estações finais. GPOS são assim denominados por não serem dedicados a aplicações específicas, tentando atender os processos e suas requisições da melhor forma possível, geralmente baseando-se em políticas simples para o compartilhamento de cada recurso.

Nas estações finais, em geral, recai a responsabilidade por tarefas de alto consumo de recursos em uma aplicação multimídia distribuída, como recuperação, codificação, transmissão, recepção, decodificação e apresentação das informações. Porém, a simplicidade de gerenciamento de recursos dos GPOS inviabiliza o suporte adequado a tais tarefas,

principalmente quando elas coexistem com outras aplicações. Na realidade, os sistemas operacionais atuais são incapazes de atingir o comportamento predizível requerido e compartilhar os recursos que gerenciam, com justiça e garantias, entre as aplicações.

Vários são os problemas encontrados na gerência de recursos de um GPOS que podem tornar inviável sua utilização em aplicações multimídia. A começar por seu subsistema de escalonamento da CPU que, em geral, recorre ao uso de prioridades para privilegiar a execução de algumas aplicações em detrimento de outras. As aplicações com maiores prioridades (por exemplo, aplicações de tempo real), somente são interrompidas por vontade própria ou para operações de entrada e saída. Esse comportamento pode ser considerado injusto, por permitir a inanição de outros processos de menor prioridade.

Por sua vez, os subsistemas de rede dos GPOS normalmente implementam o recebimento de pacotes orientado por interrupções (de hardware e de software), o que pode causar anomalias no escalonamento de processos, já que a manipulação das interrupções possui as mais altas prioridades de uso da CPU. Como conseqüência, um processo em execução com alta prioridade pode ser interrompido para o tratamento do recebimento de dados destinados a um processo de prioridade mais baixa (inversão de prioridades). O problema se agrava pelo fato do tempo de CPU utilizado nessa manipulação ser geralmente atribuído ao processo interrompido, levando a uma contabilidade equivocada no uso do recurso. A inversão de prioridades pode também ocorrer no envio de pacotes que, em geral, é implementado por meio de chamadas de sistema. Em núcleos monolíticos não-preemptivos, que ainda são comuns atualmente, se um processo de alta prioridade precisa ser escalonado para a execução no momento em que a pilha de protocolos executa um envio de dados, ele deve esperar o término da execução da chamada de sistema para então ganhar a posse da CPU.

Um último ponto crítico reside nas filas de transmissão de pacotes, compartilhadas entre as aplicações. Mesmo quando há meios para a classificação e priorização, tais mecanismos são subutilizados. Os mecanismos de controle de admissão, seja de CPU, filas de pacotes ou de outro recurso qualquer, quando existem, são integrados à política de escalonamento. De qualquer forma, não há a orquestração de recursos necessária, ou seja, os recursos são tratados de forma isolada sem que possíveis relações de dependência e sejam consideradas.

Uma funcionalidade desejável em sistemas operacionais com suporte a QoS é a adaptabilidade para a provisão de novos serviços, no intuito de acompanhar a evolução das aplicações e de suas necessidades. Conforme mencionado em [Kosmas97], adaptações a novas demandas de QoS devem ocorrer, idealmente, por meio de pequenas modificações na infra-estrutura de comunicação e processamento. Contudo, os GPOS não foram projetados para suportá-las em tempo de operação.

Os diversos trabalhos relacionados a QoS em sistemas operacionais se dedicam à extensão de sistemas já existentes ou ao desenvolvimento de novos sistemas focados no gerenciamento adequado dos recursos. Observa-se que as extensões atuam sobre subsistemas específicos (CPU, filas de pacotes, pilha de protocolos, etc.), isoladamente, não representando uma solução completa para o problema. Os novos sistemas também deixam alguns mecanismos importantes de lado, como o controle de admissão e a adaptação de serviços, que são alvos de discussão neste artigo.

A arquitetura QoSOS [Moreno 2003] aborda o problema de provisão de QoS de forma abrangente e genérica, definindo com detalhes os mecanismos necessários para seu suporte. Baseado nessa arquitetura, este artigo apresenta uma proposta para solução de alguns

problemas encontrados nas implementações existentes. Os principais conceitos introduzidos são a orquestração de recursos, o gerenciamento baseado em árvore de recursos virtuais e o mecanismo de adaptação do sistema. Tais funcionalidades são apresentadas na Seção 2. A Seção 3 apresenta o primeiro protótipo derivado da arquitetura, o sistema QoSOSLinux. Tal protótipo faz uso e aperfeiçoa algumas soluções desenvolvidas por terceiros para o sistema Linux, servindo apenas como ponto de partida para a validação de alguns conceitos definidos na arquitetura QoSOS. A Seção 4 descreve os trabalhos relacionados da área, evidenciando as principais contribuições da arquitetura proposta. Por fim, a Seção 5 se dedica às conclusões e considerações finais.

2. A arquitetura QoSOS

O desenvolvimento da arquitetura QoSOS [Moreno 2003] baseou-se na análise de algumas soluções apresentadas na literatura e na percepção de semelhanças funcionais entre elas. A arquitetura QoSOS permite reutilizar funções comuns e definir uma organização interna equivalente nos diferentes sistemas, facilitando a ação de mecanismos de orquestração de recursos fim-a-fim. A arquitetura foi definida a partir da especialização e extensão dos *frameworks* para provisão de QoS fim-a-fim em ambientes genéricos de processamento e comunicação, apresentados em [Gomes 2001]. Particularmente, QoSOS demonstra como esses *frameworks* podem ser especializados para acomodar técnicas que visem a provisão de QoS especificamente em sistemas operacionais.

2.1 - Descrição geral da arquitetura

Os frameworks foram subdivididos em quatro conjuntos, de acordo com a funcionalidade que representam na arquitetura: *Framework de Parametrização de Serviços*; *Frameworks de Compartilhamento de Recursos*; *Frameworks de Orquestração de Recursos*; *Frameworks de Adaptação de Serviços*. Maiores detalhes Uma documentação completa sobre cada *framework* pode ser encontrada em [Moreno 2002].

O *framework para parametrização de serviços* modela as estruturas de dados responsáveis por definir um esquema genérico de parâmetros de caracterização de serviços, agrupados em categorias de serviço. O *framework* permite a construção de uma hierarquia de parâmetros pela qual se faz possível a definição de informações sobre o serviço em qualquer nível de abstração. Assim, quando uma solicitação de serviço é feita, a aplicação informa parâmetros de caracterização da carga e da QoS em alto nível, os quais devem ser traduzidos em parâmetros de mais baixo nível até que se atinja uma descrição das necessidades de uso de um recurso em particular. Por exemplo, uma especificação como “vídeo com qualidade de TV” pode ser mapeada para a descrição “vídeo com 30fps, no formato SIF”. Ou, ainda, para uma taxa de utilização de 20% da CPU e uma largura de banda de 242Mbps no canal de comunicação.

Os *frameworks para compartilhamento de recursos* se baseiam no conceito de recurso virtual para modelar os mecanismos de escalonamento e de alocação de recursos. Recursos virtuais são parcelas de utilização de um ou mais recursos reais distribuídas entre os fluxos (de dados, de instruções, etc.) submetidos pelos usuários. A estrutura de gerenciamento de recursos virtuais é organizada de maneira hierárquica, através das árvores de recursos virtuais. Por ser um dos principais conceitos introduzidos pela arquitetura, a árvore de recursos virtuais receberá atenção especial neste artigo, discutida em detalhes na Seção 2.2.

Em meio à diversidade de recursos controlados pelo sistema operacional, observa-se que os serviços a serem oferecidos forçam interdependências no uso desses recursos, que devem ser consideradas de forma integrada no momento da configuração dos mecanismos individuais de escalonamento e de alocação. Daí vem o conceito de orquestração de recursos, que define a divisão na responsabilidade de provisão da QoS entre os subsistemas participantes do serviço. A orquestração de recursos é apresentada em profundidade na Seção 2.3. Na arquitetura QoSOS, a modelagem da orquestração de recursos se subdivide em dois *frameworks* distintos. O *framework para negociação de QoS* modela os mecanismos de admissão, negociação e mapeamento que operam durante as fases de solicitação e estabelecimento de serviços. Já o *framework para sintonização de QoS* modela os mecanismos de sintonização e monitoração que atuam na fase de manutenção dos serviços.

O *framework para adaptação de serviços* descreve "meta-mecanismos" que automatizam a adaptação do sistema a novos serviços ou a novas políticas de provisão de QoS, observando questões como manutenção de consistência e restrições relacionadas à segurança do sistema.

2.2 – Árvore de Recursos Virtuais

Para facilitar o emprego de vários algoritmos de escalonamento, classificação e controle de admissão sobre um mesmo recurso e, assim, oferecer um conjunto amplo e flexível de serviços em um mesmo sistema, os recursos virtuais são dispostos em uma estrutura genérica, chamada de árvore de recursos virtuais. O conceito de árvore de recursos virtuais denota abstratamente a divisão hierárquica de parcelas de utilização de um ou mais recursos, que são associadas a mecanismos de escalonamento e alocação. O nó raiz de uma árvore é chamado de escalonador do recurso raiz, podendo esse recurso raiz corresponder a: um recurso real (como banda passante, CPU, memória etc.); um conjunto de recursos reais ou virtuais analisado como um recurso único; um conjunto de recursos reais ou virtuais que são gerenciados isoladamente.

De qualquer forma, o papel do escalonador de recurso raiz é distribuir parcelas de utilização de seu respectivo recurso raiz entre seus nós filhos. Cada um desses nós, por sua vez, distribui sua própria parcela de utilização do recurso raiz entre seus nós filhos e assim sucessivamente até os nós folhas, que são os recursos virtuais alocados a um serviço ou aplicação. Uma árvore de recursos virtuais é exemplificada pela Figura 1.

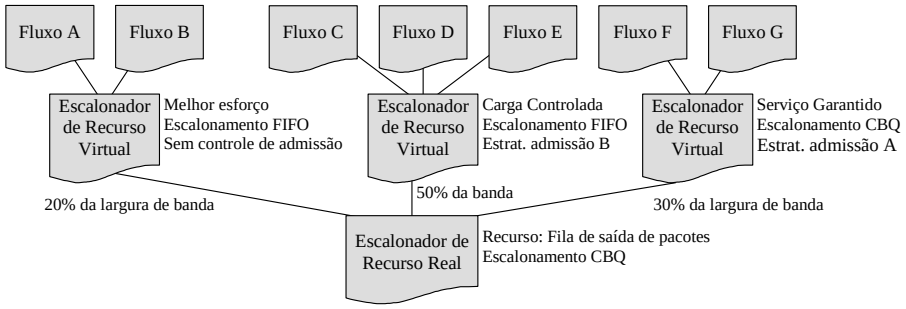

Figura 1 - Exemplo de árvore de recursos virtuais, para o recurso fila de pacotes de saída

Cada escalonador na árvore associa-se a um conjunto de políticas de QoS que definem sua categoria de serviço, estratégia de escalonamento, estratégia de admissão de novos recursos virtuais e os métodos para a criação dos recursos virtuais admitidos, como já mencionado. A seqüência de passos para a alocação de recursos é descrita na Seção 2.3. No caso mais simples, quando o recurso raiz de uma árvore corresponde a um único recurso real ou a um

conjunto de recursos reais ou virtuais analisado como um recurso único, a admissão de novos fluxos do usuário é dita primitiva, porque é feita apenas por meio da criação de recursos virtuais diretamente nos escalonadores dessa árvore. Quando a raiz representa um conjunto de recursos gerenciados isoladamente, a admissão de fluxos é repassada recursivamente para as árvores de recursos virtuais de cada recurso do conjunto, até que uma árvore de recursos virtuais com admissão primitiva seja alcançada. Nesse caso, houve uma "orquestração dos recursos".

Oferecendo diferentes formas de escalonamento do recurso para cada categoria de serviço, a árvore de recursos virtuais evita os problemas do uso de um único algoritmo baseado em prioridades presentes no escalonamento de CPU dos GPOS, ou das filas compartilhadas FIFO no envio de pacotes de rede. Além disso, com as sucessivas divisões em parcelas de utilização do recurso, fica garantida a proteção entre as categorias de serviço. Ou seja, a sobrecarga no recurso gerada por aplicações de uma mesma categoria de serviço (talvez porque as políticas de QoS associadas realmente permitam isso), não acarretará em violação da QoS de aplicações de outras categorias. O compartilhamento justo do recurso com proteção entre categorias de serviço são benefícios que compensam o custo computacional inserido pelo escalonamento hierárquico. Nota-se, no entanto, que não se trata apenas de se oferecer escalonamento hierárquico de recursos, mas também prover, o controle de admissão em cada uma das classes.

A estrutura dinâmica da árvore de recursos virtuais simplifica a implementação de mecanismos de adaptação de serviços. Novos serviços são adicionados por meio da criação de novos escalonadores de recursos virtuais nas árvores envolvidas, e da introdução de suas novas políticas de QoS. A adaptação de um serviço existente se reduz à troca das políticas de QoS associadas ao escalonador que se quer adaptar.

O dinamismo da árvore também beneficia a correta contabilidade e uso apropriado do recurso, apontados como pontos críticos nos GPOS. Conforme a necessidade, fluxos podem migrar entre escalonadores de recursos e, conseqüentemente, entre categorias de serviço. Dessa forma, em um cenário em que um mesmo fluxo de programa é utilizado por diversas aplicações (seja essa parcela do recurso utilizada no domínio do núcleo ou do usuário), o fluxo pode ser transferido para ser escalonado no contexto da categoria de serviço da aplicação para a qual ele está dedicando seu trabalho. Por exemplo, no caso do tratamento das interrupções de recepção de pacotes de rede, se o processamento da pilha for executado por uma mesma *thread* de núcleo para todos os pacotes, essa *thread* pode ser associada, sob demanda, ao escalonador da categoria de serviço da aplicação a que se destinam os pacotes. Esse problema foi levantado inicialmente por Banga (1999), que propôs uma solução semelhante, discutida na Seção 4.

2.3 – Orquestração de recursos

Conforme mencionado anteriormente, orquestrar recursos significa dividir a responsabilidade da QoS entre os subsistemas envolvidos na provisão do serviço. Em termos de árvores de recursos virtuais, a admissão de fluxos é repassada recursivamente para as árvores de recursos virtuais de cada subsistema, até que uma árvore de recursos virtuais com admissão primitiva seja alcançada.

Em um sistema operacional, aplicações que desejam solicitar serviços com QoS devem realizar um pedido de admissão ao controlador de admissão da árvore de recursos virtuais representando todo o sistema. Potencialmente, tal requisição demandará a reserva de diversos

recursos, o que obriga o acionamento de um agente negociador de QoS. O negociador deve identificar todos os subsistemas envolvidos no fornecimento do serviço (CPU, subsistemas de rede etc.) e, utilizando-se da estratégia de orquestração a ele associada, distribuir entre esses subsistemas (de fato, as árvores de recursos virtuais que os representam) a parcela de responsabilidade sobre a provisão da QoS especificada.

A partir das parcelas de responsabilidade atribuídas a cada recurso ou subsistema (recursos virtuais), são acionados os mecanismos de mapeamento, consistindo na tradução da categoria de serviço (e dos valores dos parâmetros associados) especificada na solicitação do serviço para as categorias de serviço (e valores de parâmetros associados) específicas relacionadas diretamente com a capacidade de operação dos recursos orquestrados. O mecanismo de controle de admissão de cada recurso virtual deve, então, ser recursivamente acionado a fim de verificar a viabilidade de aceitação do novo fluxo. Esse procedimento recursivo segue até que uma árvore de recursos virtuais com admissão primitiva seja alcançada. Em cada nível de recursão, cada controlador de admissão pode responder de forma afirmativa a solicitação, retornando um identificador de pré-reserva ao negociador. Se todos os controladores de admissão responderem afirmativamente, um outro identificador de pré-reserva será gerado pelo negociador e entregue ao nível superior de recursão, até que a admissão na árvore de recursos virtuais representando todo o sistema operacional é efetuada. Caso algum controlador de admissão de um dos recursos virtuais orquestrados responda negativamente, a requisição do serviço pode ser imediatamente negada ou o mecanismo de negociação pode reiniciar o processo, com nova redistribuição das parcelas de responsabilidade.

Nota-se que nenhuma reserva efetiva é realizada até que a aplicação confirme a solicitação do serviço. Na realidade, até então os novos recursos virtuais admitidos não foram criados, porém sua parcela de uso do recurso passa a ser considerada para outras admissões concorrentes, por um certo período de tempo. Esse esquema de admissão em dois passos, do tipo *admit/commit* permite um suporte adequado a esquemas de negociação distribuída, e evita a ocorrência de deadlocks na reserva dos recursos.

A confirmação da solicitação é feita quando a aplicação informa o identificador gerado anteriormente ao controlador de admissão do sistema operacional. O negociador é invocado novamente para acionar os mecanismos de criação de recursos virtuais de cada um dos subsistemas participantes da orquestração. Com a criação bem sucedida dos recursos virtuais, implicitamente está estabelecido o contrato de serviço.

3. O protótipo QoSOSLinux

QoSOSLinux é uma extensão ao sistema operacional Linux instanciada a partir da arquitetura QoSOS [Moreno 2003]. O objetivo que se propõe é adicionar progressivamente as funcionalidades contempladas pela arquitetura a um GPOS popular. Como pôde ser observado ao longo deste artigo, a arquitetura é extensa e rica em detalhes, o que demanda grande esforço no desenvolvimento de protótipos. Adotou-se, assim, uma estratégia de planejamento que, inicialmente, prevê a utilização de alguns mecanismos de gerência de recursos já desenvolvidos por terceiros para a validação dos modelos de orquestração, compartilhamento e alocação de recursos, bem como o suporte a adaptabilidade de serviços. Gradativamente, os mecanismos estão sendo substituídos por implementações nativas em total conformidade com a arquitetura.

3.1. QoSOSLinux v0.1

De acordo com o planejamento mencionado, esta seção apresenta o primeiro protótipo QoSOSLinux, que utiliza e estende os mecanismos de controle de tráfego do Linux (LinuxTC [Almesberger 1998]) para o gerenciamento das filas de transmissão de pacotes. Para o escalonamento de processos, foi introduzido o suporte ao escalonador dinâmico de tempo real DSRT2 [Nahrstedt 1999]. Detalhes da adequação dos dois trabalhos à arquitetura QoSOS são descritos nas próximas seções. Adicionalmente, foi aproveitada a extensão LRP (*Lazy Receiver Processing* [Druschel 1996]), para maior justiça no processamento da recepção de pacotes de rede, evitando anomalias no escalonamento de processos, como também já mencionado.

3.1.1 - Incorporação do LinuxTC

As versões mais recentes do núcleo do sistema Linux oferecem um grande conjunto de funções de controle de tráfego de rede, capazes de configurar o compartilhamento das filas de envio de pacotes das interfaces de rede. O controle é composto pelos seguintes componentes conceituais: disciplinas de escalonamento, classes e filtros de classificação e policiamento. Cada interface de rede tem a ela associada uma fila, que por sua vez tem associada uma classe e sua política de escalonamento. O escalonamento pode definir novas classes, cada uma com política distinta de escalonamento, em uma estrutura de escalonamento hierárquico, onde as classes folha na hierarquia são responsáveis pelo escalonamento de pacotes a elas pertencentes. Filtros de classificação são utilizados para distinção de um pacote e sua distribuição nas diversas classes, as quais podem ter associadas ações de policiamento e moldagem de tráfego.

A árvore de recursos virtuais, introduzida na Seção 2.2, é genérica o suficiente para abranger o controle de tráfego do Linux, com pequenas modificações internas. Como o LinuxTC não foi projetado para permitir que aplicações e protocolos de negociação configurem as características de desempenho da comunicação pela rede, os esforços de desenvolvimento do sistema QoSOSLinux concentraram-se, inicialmente, na definição de uma API, em um projeto internacional de código aberto chamado TCAPI [Olshefski01], com apoio da IBM.

LinuxTC não possui qualquer mecanismo de controle de admissão, o que permite a ocorrência de sobrecarga nas disciplinas de enfileiramento e nega a possibilidade de garantias de serviço. Um controlador de admissão de fluxos foi então adicionado ao sistema para contornar o problema. O controlador foi idealizado de forma a prover adaptabilidade, permitindo que as estratégias de admissão sejam modificadas em tempo de execução. Para isso, o núcleo do Linux foi modificado de forma que o subsistema de gerência de módulos de núcleo seja capaz de manipular a substituição e introdução de estratégias de admissão.

3.1.2 - Incorporação do DSRT2

O escalonador de CPU do sistema Linux é baseado apenas em prioridades, não oferecendo suporte adequado a aplicações multimídia. Para incorporar um escalonamento baseado no tempo à CPU, optou-se pela utilização do DSRT2. Apesar de suas limitações, como será visto, a integração do DSRT2 permitirá o suporte, embora limitado, a aplicações de tempo real suave, e também o teste da orquestração de recursos, discutido na próxima seção.

O escalonador DSRT2 é um servidor de nível de usuário capaz de oferecer garantias a aplicações de tempo real suave, através de atualizações nas prioridades dos processos no escalonador do núcleo do sistema. O escalonador é uma implementação independente de

plataforma, caracterizando sua flexibilidade como uma de suas maiores virtudes. As aplicações podem requisitar suas necessidades de utilização da CPU de diversas maneiras, se enquadrando em uma das classes de aplicação definidas. O servidor é inicializado com um parâmetro que define qual a percentagem de CPU estará disponível para aplicações de tempo real, sendo o restante destinado a aplicações de melhor esforço. Cada requisição de processamento em tempo real é colocada em uma tabela que descreve um plano de execução, cuja ordem de despacho é regida por um algoritmo semelhante ao EDF (Earliest Deadline First [Liu 1973]).

O servidor de escalonamento foi facilmente configurado para a utilização no protótipo QoSOSLinux, por ser um programa de nível de usuário, que não demanda modificações no núcleo e por disponibilizar uma API de configuração orientada a objetos (C++ e Java). Suas classes de aplicação foram mapeadas diretamente para categorias de serviço da arquitetura QoSOS. O programa mostrou-se eficiente para lidar com aplicações de tempo real quando há facilidade de definição de parâmetros como período e tempo de execução. Para os casos em que tais medidas são de difícil percepção, o servidor disponibiliza funções de teste (*probing*) para inferir os parâmetros a serem fornecidos.

No DSRT2, o controle de admissão é integrado ao escalonador, obrigando que admissão e reserva sejam feitas em apenas um passo. Essa incompatibilidade com a arquitetura QoSOS foi contornada pela modificação interna do DSRT2, introduzindo o esquema *admit/commit* ao processo de solicitação de QoS. Observou-se que o sistema não é capaz de oferecer suporte a aplicações de tempo real severo, que não toleram variações de retardo, devido ao fato de que o controle em nível de usuário não é capaz de oferecer granularidade fina nas garantias de QoS.

Por fim, constatou-se que não é possível oferecer proteção entre as categorias de serviço já que todas são regidas por um único algoritmo de escalonamento. A diferença entre categorias somente é notada nos parâmetros distintos de requisição do serviço. Quando ocorre uma violação no uso da CPU, pode haver a degradação da QoS de todos os processos de tempo real. Além disso, não há mecanismos de adaptação de serviços. Para minimizar o problema, o DSRT2 conta apenas com estratégias de sintonização para corrigir parâmetros de escalonamento na presença de comportamento inadequado de processos. Optou-se por não despender esforços na construção da árvore de recursos virtuais internamente ao DSRT2, devido às restrições do programa impostas pelo seu processamento em espaço do usuário.

3.1.3 – Orquestração entre DSRT2 e LinuxTC

Para atender a uma aplicação multimídia distribuída, QoSOSLinux disponibiliza o controlador de admissão QoSOSLinux, responsável pela árvore de recursos virtuais que representa todo o sistema operacional. O pedido de admissão é repassado ao negociador QoSOSLinux, que identifica os subsistemas participantes capazes de oferecer garantias de QoS, inicialmente o escalonamento de processos (DSRT2) e de pacotes (LinuxTC). Baseado em sua estratégia de orquestração, o negociador deverá dividir a parcela de responsabilidade da QoS entre esses subsistemas. Deve ser usada uma estratégia de orquestração que leva em conta o fato de que, no Linux, o processamento da pilha de protocolos está no contexto de uma chamada de sistema (seja no envio, seja na recepção estendida pelo LRP, anteriormente comentada). Pode-se dizer, então, que o processamento da aplicação para manipular os dados e o processamento da pilha para a comunicação estão em um mesmo contexto de execução, ou seja, não há *threads* separadas para a execução das tarefas da aplicação e da pilha de protocolos.

A estratégia de orquestração de recursos pode, portanto, distribuir as responsabilidades da seguinte forma. Os requisitos para reserva da CPU, para *thread* da aplicação, a serem enviados ao controlador de admissão do DSRT2 equivalem à soma entre os requisitos necessários para a própria *thread* manipular os dados e para a pilha de protocolos no núcleo processar os pacotes, de acordo com a caracterização de tráfego especificada. Já os requisitos de banda passante a serem encaminhados ao controlador de admissão do LinuxTC, para a fila de pacotes de saída, equivalem aos requisitos determinados pelo protocolo de negociação de rede para a estação.

Como é muito difícil a inferência de qual deve ser o processamento a ser requisitado para a pilha de protocolos de comunicação, foi implementado no QoSOSLinux um mecanismo de calibração de QoS para o subsistema de rede, executado a cada inicialização do sistema. O mecanismo funciona como o *probing* do DSRT2, mas para dados genéricos. Para vários tipos de carga, é medida a necessidade de CPU no processamento da pilha e os resultados são armazenados. No momento de uma solicitação de serviço, o negociador pode se basear em tais informações para inferir sobre o requisito de processamento a ser utilizado pela estratégia de orquestração. Isso evita que as aplicações sejam obrigadas a fazer testes de medição antes de iniciar a comunicação dos dados, garantindo maior transparência no processo de admissão.

4. Trabalhos relacionados

Os trabalhos relacionados à provisão de QoS em sistemas operacionais incluem desde propostas de extensão para GPOS a projetos de novos sistemas operacionais, com foco sobre mecanismos de gerenciamento de recursos. Neste artigo não serão citados todos os trabalhos já analisados por falta de espaço, mas outras discussões podem ser encontradas em [Moreno 2002, 2003].

O modelo QualMan [Nahrstedt 1999], que inclui o DSRT2 como servidor de escalonamento de processos, define vários mecanismos para a negociação de QoS em sistemas operacionais, sendo que muitos deles possuem representantes análogos na arquitetura QoSOS. Porém, toda a arquitetura QualMan possui as desvantagens já citadas anteriormente para o DSRT2, não observadas na arquitetura QoSOS.

Banga (1999) aborda o problema da contabilidade equivocada de uso dos recursos por núcleos monolíticos e propõe o modelo de *resource containers* principalmente para a correta atribuição do tempo de uso da CPU ao processo responsável pelo processamento realizado. Como demonstrado na Seção 2.2, a árvore de recursos virtuais permite funcionalidade análoga, pela migração dos fluxos entre as categorias de serviço. Pode-se considerar que o modelo de árvore de recursos virtuais vai além dos *resource containers* por permitir a definição e o uso de diferentes políticas de QoS entre os nós da hierarquia.

5. Conclusões e trabalhos futuros

Neste trabalho foram apresentados detalhes da arquitetura QoSOS para provisão de QoS adaptável em sistemas operacionais. Foram ressaltadas as funcionalidades da arquitetura que garantem a correção dos problemas encontrados em sistemas operacionais de propósito geral nesse contexto. O planejamento e desenvolvimento do protótipo QoSOSLinux foi apresentado, bem como detalhes da implementação de uma versão preliminar para validação dos mecanismos definidos pela arquitetura. Demonstra-se, assim, como uma extensão de um GPOS pode ser elaborada, de forma a prover mecanismos de provisão de QoS frente a todas as deficiências que se apresentam na implementação

de tal tipo de serviço. Devido às restrições de espaço neste documento, a análise dos resultados preliminares do sistema são apresentados à parte, disponíveis em http://www.telemidia.puc-rio.br/~moreno/qososlinux/0.1/analise.html.

As próximas versões do QoSOSLinux contornarão certas deficiências do LinuxTC com relação à adaptação de serviços por meio da remodelagem do subsistema em total conformidade com a árvore de recursos virtuais. Um novo escalonador de processos adaptável implementado no núcleo, será introduzido, baseado no conceito de árvore de recursos virtuais. Gradativamente, outros recursos serão incluídos no suporte a QoS, como memória, sistema de paginação, etc.

Referências

Almesberger, W. "Linux network traffic control". Implementation Details ftp://icaftp.epfl.ch/pub/people/almesber/pub/tcio-current.ps.gz, 1999.

Banga, G., Druschel, P., Mogul, J. "Resource containers: A new facility for resource management in server systems". In Proceedings of the Third Symposium on Operating Systems Design and Implementation (OSDI'99), New Orleans, 1999.

Druschel, P., Banga, G. "Lazy receiver processing (LRP): a network subsystem architecture for server systems". Proceedings of 2nd Symposium on Operating System Design and Implementation (OSDI'96), p. 261-275, 1996.

Gomes, A.T., Colcher S., Soares, L.F. "Modeling QoS Provision on Adaptable Communication Environments", Proceedings of the IEEE International Communication Conference (ICC2001), Helsinki, Finland, 2001.

Kosmas, N., Turner, K. "Requirements for service creation environments". International Workshop on Applied Formal Methods in System Design, p. 133-137, 1997

Liu, C.; Layland, J. "Scheduling algorithms for multiprogramming in a hard-real-time environment". Journal of the ACM, v. 20, p. 46-61, 1973.

Moreno, M. "Um framework para provisão de QoS em sistemas operacionais". M.Sc. Thesis, Pontifical Catholic University of Rio de Janeiro, 2002.

Moreno, M., Soares Neto, C., Gomes, A.T., Colcher, S., Soares, L.F. "QoSOS: An adaptable architecture for QoS provisioning in network operating systems". Journal of the Brazilian Telecommunications Society, Special Issue, v.18, n.2, p.118-131, 2003.

Nahrstedt, K., Chu, H., Narayan, S. "QoS-aware Resource Management for Distributed Multimedia Applications", Journal on High-Speed Networking, Special Issue on Multimedia Networking, v. 8, n. 3-4, 1998, p.227-255, 1999.

Olshefski, D. "Notes on linux network QoS – TCAPI version 1.0". Work in progress. ftp://www-126.ibm.com/pub/tcapi/tcapi.tar.gz, 2001

Pree, W. "Design patterns for object-oriented software development". Addison Wesley, 1995.