Atendendo aos requisitos de QoS de aplicações multimídia através da especialização de serviços do sistema operacional

Luciano Porto Barreto

Laboratório de Sistemas Distribuídos - LaSiD
Universidade Federal da Bahia - UFBA
Av. Adhemar de Barros, S/N, Campus de Ondina
Salvador-BA, CEP 40.170-110

lportoba@ufba.br

Resumo. *Atender às necessidades de aplicações multimídia requer freqüentemente a utilização ou desenvolvimento de sistemas de execução específicos que forneçam o nível de qualidade de serviço esperado por tais aplicações. Este artigo apresenta uma infraestrutura que pode ser usada por programadores de aplicações multimídia para adaptar a política de escalonamento de processos do sistema operacional com o objetivo de melhorar o desempenho de suas aplicações. Apresentamos o desenvolvimento de escalonadores de processos dedicados à gestão de aplicações multimídia utilizando esta infraestrutura e o seu desempenho na gestão de duas aplicações multimídia amplamente utilizadas.*

Abstract. *Addressing the needs of multimedia applications frequently requires the use or development of specialized run-time systems that are able to provide the quality of service required by these applications. This paper presents a framework that can be used by multimedia developers to adapt the process scheduling policy to improve the performance of their applications. We present the development of specialized process schedulers for multimedia applications and the performance of two widely used multimedia applications under these schedulers.*

1 Introdução

Nos últimos anos aplicações multimídia como *streaming* de áudio e videoconferência tornaram-se comuns em computadores pessoais devido à crescente evolução tecnológica do hardware e do surgimento de novas aplicações na área. Entretanto, a utilização de tais aplicações em sistemas operacionais convencionais como Linux e Windows NT é por vezes prejudicada pois estes últimos não foram concebidos para atender aos requisitos de aplicações multimídia, mas sim para suportar uma carga diversa de processos a exemplo de editores de texto e aplicações de cálculo. Como resultado, certas aplicações multimídia apresentam um desempenho ou qualidade de serviço insatisfatórios podendo, inclusive, impedir sua execução.

Um dos principais problemas na execução de aplicações multimídia reside em garantir que o suporte de execução (por exemplo, máquina virtual, *middleware* ou o próprio sistema operacional) forneça a qualidade de serviço necessária ao bom funcionamento da aplicação. Por exemplo, aplicações de vídeo requerem uma taxa mínima de exibição de quadros e um limite no atraso do tratamento de quadros sucessivos.

O desenvolvedor de uma aplicação multimídia deve então traduzir os requisitos de qualidade de serviço da aplicação em métricas que possam ser utilizadas pelo suporte de execução para fornecer tais garantias. Entretanto, mesmo quando tais garantias estão disponíveis, tal tradução é difícil de ser realizada devido à falta de uma interface adequada do sistema operacional. Esta interface se resume nos sistemas operacionais convencionais, tais como Windows NT e Linux, à escolha entre níveis de prioridade, o que não fornece nenhuma informação específica sobre os requisitos da aplicação ao sistema operacional, mais sim um parâmetro relativo em relação a outras aplicações. A incapacidade dos sistemas operacionais em atender aos requisitos de certas aplicações impulsionou o desenvolvimento de sistemas operacionais e *middleware* dedicados à multimídia. Alguns exemplos incluem 2K [15], Nemesis [17], TAO [27], QLinux [32] e VINO [28]. Infelizmente tais serviços ou sistemas são difíceis de serem utilizados ou adaptados para outros contextos de execução.

Fornecer garantias temporais a uma aplicação requer uma reserva de recursos no nível do sistema operacional, em especial, em termos de tempo de processador. É necessário que o escalonador de processos considere as necessidades da aplicação multimídia para garantir um serviço adequado. De modo a suprir essa necessidade, diversas políticas de escalonamento especializadas têm sido propostas no contexto de sistemas multimídia [7, 8, 10, 20, 23, 31, 34, 35]. Na ausência de sistemas operacionais com mecanismos adequados às necessidades de suas aplicações, o desenvolvedor multimídia é levado a desenvolver novos mecanismos de reserva de recursos que, por vezes, são implementados diretamente no núcleo do sistema operacional. Entretanto, o desenvolvimento de tais mecanismos é complexo pois requer um conhecimento profundo da estrutura e funcionamento interno do sistema operacional.

Recentemente desenvolvemos a plataforma Bossa [2, 3] cujo objetivo é facilitar o desenvolvimento de escalonadores de processos e permitir sua integração de forma automática no núcleo do Linux. Neste artigo, mostramos como os escalonadores desenvolvidos nessa plataforma podem ser usados e adaptados por desenvolvedores de multimídia para atender aos requisitos de suas aplicações. Nossa solução dispensa a complexidade da utilização de um *middleware* ou sistema operacional específico e permite que programadores de aplicação sem experiência em programação de sistemas operacionais possam adaptar tanto os mecanismos do sistema operacional quanto suas aplicações a fim de garantir um nível adequado de qualidade de serviço. Apresentamos o desempenho de duas aplicações multimídia largamente difundidas (o Berkeley MPEG player [5] e o `mplayer` [19]) quando utilizadas sob o controle de escalonadores dedicados desenvolvidos na plataforma Bossa. Descrevemos as modificações realizadas nas duas aplicações e o desempenho obtido com dois escalonadores de processos : (i) baseado na noção de *progresso* da aplicação de vídeo e (ii) um escalonador que efetua uma reserva de recursos.

O restante desse artigo está estruturado da seguinte forma. A seção 2 apresenta a plataforma Bossa, que permite o desenvolvimento e integração de escalonadores de processos no núcleo do Linux. Em seguida, as seções 3 e 4 ilustram a utilização prática dessa plataforma através da adequação e análise de desempenho de duas aplicações multimídia. Na seção 5 é feita uma discussão sobre os trabalhos relacionados e na seção 6 apresentamos algumas considerações finais e possíveis extensões deste trabalho.

2 A plataforma Bossa

Bossa é uma plataforma voltada ao desenvolvimento de políticas de escalonamento de processos. A plataforma Bossa possui dois componentes principais : (i) uma linguagem de programação dedicada que fornece um conjunto de abstrações específicas com o objetivo de facilitar o desenvolvimento de políticas de escalonamento e (ii) um suporte de execução que permite a integração automática de uma política de escalonamento desenvolvida na linguagem Bossa ao núcleo do sistema operacional. Bossa foi implementada no sistema operacional Linux e está disponível na versão 2.4 do núcleo.

A Figura 1 descreve o processo de desenvolvimento através da plataforma Bossa. O desenvolvedor escreve uma política de escalonamento utilizando as abstrações fornecidas pela linguagem dedicada. A partir de uma especificação Bossa, o compilador verifica a coerência da política de escalonamento [16] e gera o escalonador de processos em código C que deve ser integrado ao núcleo do sistema operacional e um arquivo contendo os *stubs* das funções definidas pela interface do escalonador a exemplo da função `setpriority` no Linux. O arquivo contendo os *stubs* podem ser compilados com aplicações convencionais para que estas possam utilizar as funções exportadas pela interface do escalonador.

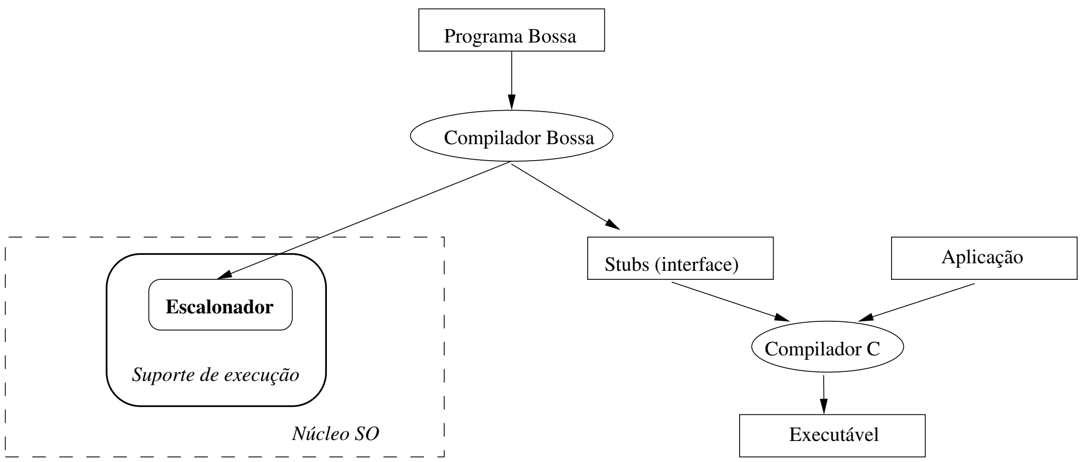

Figura 1: Desenvolvimento de escalonadores de processo e aplicações na plataforma Bossa

Um escalonador escrito na linguagem Bossa é composto de três partes : (i) declarações de atributos e estados de um processo, (ii) definição dos tratadores de eventos e (iii) a especificação de uma interface que permite a interação entre os processos e o escalonador. A seguir descrevemos esses componentes.

As declarações permitem a definição dos atributos dos processos, do critério de escalonamento, dos estados, variáveis e tipos enumerados. Os atributos de um processo especificam as informações associadas a cada processo como a prioridade e o quantum de execução. O critério de escalonamento define a relação de importância entre os processos (por exemplo, FIFO, maior prioridade) e serve para a escolha do novo processo a ser executado. Os estados de um escalonador definem as estruturas utilizadas para armazenar os processos como as filas de espera e variáveis.

Os tratadores de evento de um programa Bossa definem o comportamento do escalonador assim que um evento é sinalizado pelo núcleo do sistema operacional. A linguagem Bossa define um conjunto de eventos relacionados ao escalonamento de processos resumidos na Tabela 1. Um tratador de evento tipicamente move um processo entre filas de espera, para sinalizar uma mudança de estado do processo em questão. Um tratador de evento ainda pode modificar os atributos de um processo (por exemplo, atualizar a prioridade de um processo) ou variáveis globais do escalonador.

Evento	Significado
`block.*`	Bloqueio de um processo
`unblock.*`	Desbloqueio de um processo
`yield.user.*`	Liberação voluntária do processador
`system.clocktick`	Interrupção do relógio do sistema
`process.new.*`	Criação de um processo
`process.end`	Fim de um processo
`bossa.schedule`	Seleção de um novo processo

Tabela 1: Principais eventos de escalonamento da linguagem Bossa

A interface do escalonador é composta de um conjunto de funções permitindo aos processos comunicar-se com o escalonador. As funções definidas por essa interface podem ser utilizadas para modificar os atributos de um processo e para consultar ou modificar os parâmetros globais de escalonamento.

Diversas políticas de escalonamento de processos foram desenvolvidas em Bossa como, por exemplo, o escalonador Linux original [4], o escalonador Linux juntamente com a política EDF, bem como políticas baseadas na noção de partição proporcional de tempo (*proportional sharing*) [22, 33] e na noção de progresso de uma aplicação [31, 34], bem como variantes baseadas em prioridades. Além disso, a linguagem e o suporte de execução permitem a construção de infraestruturas de escalonamento hierárquicas [11, 13, 25, 32] através das quais pode-se fornecer serviços de escalonamento diferenciados à classes de aplicação específicas.

2.1 Especializando escalonadores de processo e aplicações em Bossa

Do ponto de vista do desenvolvedor de aplicações, o essencial consiste em escolher um escalonador que forneça um serviço e interface adequados às necessidades de suas aplicações. Identificar tais requisitos requer o conhecimento preciso do funcionamento da aplicação. Por exemplo, uma aplicação de cálculo pode requerer uma taxa de execução de 1000 interações por segundo ao passo que uma aplicação de vídeo pode solicitar uma taxa de execução de 30 quadros por segundo. É importante notar que a especificação “30 quadros por segundo” não implica necessariamente em uma alocação fixa do tempo do processador já que o tempo de processamento necessário à (de)codificação do vídeo pode variar em função do algoritmo ou esquema de (de)codificação utilizado[1]. Em seguida, o desenvolvedor deve mapear os requisitos da aplicação aos atributos associados a cada processo e definidos pela interface do escalonador, tais como periodicidade de execução, tempo máximo de execução dentro de cada período e taxa de execução.

Uma das vantagens da utilização da plataforma Bossa é que além de poder escolher entre diferentes políticas de escalonamento, os desenvolvedores podem adaptar as políticas de escalonamento fornecidas pelo sistema operacional no intuito de melhorar o desempenho de aplicações específicas. Isso decorre da facilidade em modificar as políticas de escalonamento através da linguagem dedicada, o que dispensa um conhecimento específico do funcionamento e dos detalhes de implementação do sistemas operacional. Assim, toda a complexidade relativa aos detalhes de programação do sistema operacional é capturada pelas abstrações da linguagem.

Para ilustrar uma aplicação prática da utilização da plataforma Bossa, descreveremos a seguir dois exemplos de aplicações multimídia (o Berkeley MPEG player e o `mplayer`) sendo executadas sob a supervisão de dois escalonadores desenvolvidos em Bossa a fim de melhorar a qualidade de serviço dessas aplicações. Os escalonadores foram desenvolvidos a partir da adaptação da política de escalonamento do Linux implementada em Bossa. Descrevemos inicialmente as modificações realizadas nas aplicações e uma visão geral sobre a estratégia de escalonamento utilizada. Em seguida, comparamos o desempenho das aplicações multimídia sob o controle do escalonador Linux original e o escalonador específico desenvolvido em Bossa.

3 Cenário I : Berkeley MPEG player com um escalonador baseado na noção de progresso

Uma abordagem emergente que tem sido utilizada para controlar a alocação de tempo concedida a uma aplicação consiste em utilizar a noção de *progresso* associada à execução da aplicação [9, 31, 34]. Esse tipo de política de escalonamento calcula o progresso de uma aplicação a partir da observação dos recursos do sistema ou de métricas de desempenho associadas à execução da aplicação. Por exemplo, um servidor HTTP poderia definir um número mínimo de requisições a serem tratadas por segundo ou uma taxa de transferência em bytes. Comparando os valores esperados aos valores atuais associados a essas métricas de desempenho, o escalonador pode fornecer uma alocação de tempo mais apropriada segundo às necessidades da aplicação.

3.1 Política de escalonamento

Nesta seção, apresentamos a implementação de uma política de escalonamento que utiliza a noção de progresso para controlar a importância do Berkeley MPEG player (`mpeg_play`) [5]. Esta política foi implementada como uma extensão do escalonador Linux desenvolvido em Bossa. A nova política baseia-se na observação de duas métricas que definem a noção de progresso da aplicação `mpeg_play` : a taxa de quadros atual (em quadros por segundo) e a taxa de quadros requerida pela aplicação (codificada no arquivo MPEG ou especificada pelo usuário através da linha de comando). A estratégia consiste

[1]Por exemplo, o padrão MPEG define três tipos de quadros, B, P e I, cujos tempos de processamento podem apresentar variações importantes.

em utilizar essas duas métricas para ajustar dinamicamente a importância da aplicação de modo que a taxa de quadros atual seja a mais próxima possível da taxa de quadros requerida. Para implementar essa política, foi introduzida uma nova classe de processos (`SCHED_PROGRESS`) às classes de processos do Linux e uma nova função na interface do escalonador. Além disso, o tratador do evento de bloqueio de processo foi modificado. Essas modificações serão descritas em seguida.

Em primeiro lugar, a aplicação `mpeg_play` informa ao escalonador que sua importância deve ser controlada em função das métricas que representam sua noção de progresso. Para tanto, a aplicação utiliza a função `set_progress()` (definida na interface do escalonador) passando dois parâmetros : a taxa de quadros requerida e o endereço da variável que contém o valor da taxa de quadros atual. A função `set_progress()` inicializa o tipo de política associada ao processo (`SCHED_PROGRESS`) e atualiza os demais atributos segundo os parâmetros informados pelo processo.

O tratador de evento referente ao bloqueio de processos (`block.*`) da política Linux foi modificado para adaptar a importância de uma aplicação que especifique métricas de progresso. Assim, cada vez que um processo da classe `SCHED_PROGRESS` bloqueia-se, o tratador de evento modificado compara a taxa de quadros esperada com a taxa de quadros atual. Se a taxa de quadros atual é inferior a taxa de quadros requerida mais um valor de tolerância, o escalonador aumenta a prioridade do processo até um valor máximo. Da mesma forma, se a taxa de quadros atual excede a taxa de quadros requerida mais um valor de tolerância, a política reduz a importância do processo.

3.2 Modificações feitas no Berkeley MPEG player

As principais modificações realizadas no Berkeley MPEG player são as seguintes:

- `main.c`: a aplicação inicializa uma estrutura que contém suas métricas de progresso : a taxa de quadros atual (contida na variável `VideoFrameRate`) e a taxa de quadros requerida representada pela variável `framerate`. Estas informações são repassadas ao escalonador através da função `set_progress`.

- `gdith.c`: Na função `ExecuteDisplay`, foi adicionada uma chamada à função de interface `bossa_sched_yield` no ponto em que a aplicação efetua o controle da taxa de exibição do vídeo, para indicar ao escalonador que a aplicação finalizou a exibição de um *stream* de vídeo.

3.3 Resultados experimentais

Os testes efetuados com o `mpeg_play` e a política de escalonamento foram realizados em um laptop Compaq Pentium III-700MHz com 256Mb de memória principal utilizando o vídeo `4Dice`[2]. Este vídeo contem 1680 quadros em formato MPEG com duração de aproximadamente um minuto. Em segundo plano, foram executados entre 0 e 10 processos realizando a mesma aplicação de cálculo (Fibonnacci recursivo).

A Figura 2 apresenta a comparação entre o desempenho do escalonador baseado na noção de progresso e o escalonador Linux original. Esses resultados mostram que até

[2]http://www.cs.indiana.edu/~hanson/4dice.mpg

a execução de cinco processos de cálculo, o escalonador baseado na noção de progresso quase consegue fornecer a taxa de quadros requerida pela aplicação. De fato, verifica-se uma ligeira degradação quando há mais de cinco processos de cálculo. Em contrapartida, o escalonador Linux é incapaz de fornecer a taxa de quadros requerida pela aplicação mesmo com apenas um único processo de cálculo. Além disso, a taxa de quadros atual degrada-se até a metade da taxa de quadros requerida com dez processos de cálculo no caso do escalonador Linux.

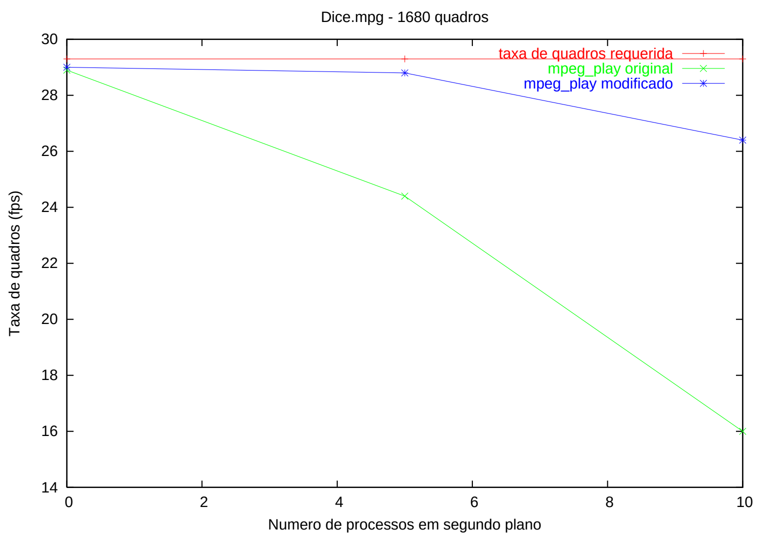

Figura 2: Taxa de quadros média do Berkeley MPEG player.

4 Cenário II : mplayer com um escalonador baseado em reserva de recursos

Uma abordagem bastante utilizada para o controle de aplicações multimídia consiste em utilizar um sistema operacional baseado na noção de reserva de tempo de processador. Uma reserva *X/Y* feita por uma aplicação especifica que *X* unidades de tempo devem estar disponíveis a cada *Y* unidades de tempo. Alguns exemplos de políticas de escalonamento baseadas em reserva de recursos incluem *Earliest Deadline First* (EDF) e *Rate Monotonic* [18] e R-EDF [35]. Em geral, estes escalonadores verificam se o processo pode ser aceito através de um teste de admissão que analisa a carga atual do sistema e a requisição de tempo especificada pela reserva do processo.

4.1 Política de escalonamento

A política de escalonamento utilizada permite que um processo com comportamento periódico reserve uma quantidade de tempo de processador dentro de cada período sucessivo. Tendo em vista que o tempo de processador requerido por uma aplicação de vídeo varia em função da complexidade de (de)codificação dos quadros, a reserva efetuada pela aplicação não é restritiva, ou seja, um processo que excede o limite de sua reserva dentro de um certo período passa a executar como um processo normal até o inicio do próximo período.

Os processos que efetuam uma reserva de tempo possuem uma prioridade mais elevada que os processos normais, mas uma prioridade inferior à prioridade dos processos de tempo-real do Linux. Entre os processos que efetuam uma reserva, aquele com o deadline mais próximo possui a prioridade mais elevada (política EDF). O período de cada processo é atualizado através de um temporizador associado ao processo. Dentro de cada período, a reserva é implementada através da alocação de tiques de relógio ao processo. Após a passagem de um tique de relógio, representada pelo evento `system.clocktick`, caso um processo do tipo EDF exceda seu número de tiques, o tratador deste evento reduz a prioridade do processo ao nível de prioridade dos processos normais até o final do período associado ao processo. Quando uma aplicação que possui uma reserva completa o seu processamento no período corrente, a aplicação deve liberar o processador utilizando a função `edf_sched_yield` definida pela interface do escalonador.

As modificações feitas à implementação Bossa do escalonador Linux são as seguintes. Foi adicionada uma nova classe de serviço, chamada `SCHED_EDF`, para processos que possuem uma reserva de processador, e o critério de escalonamento foi modificado para favorecer os processos pertencentes a esta classe sobre os processos normais (à exceção dos processos de tempo-real). Foram adicionados atributos a cada processo para armazenar os valores de deadline e informações relativas ao temporizador. O tratador do evento `unblock.timer`, que ocorre a partir de uma interrupção de relógio, foi estendido para desbloquear o processo caso este tenha liberado voluntariamente o processador e para registrar o fato de que um deadline foi violado caso o processo não tenha liberado o processador. De fato, um processo pode exceder o seu deadline em diversas situações: em execução, em espera de execução na fila de processos prontos ou na fila de processos bloqueados. Enfim, o tratador de evento `system.clocktick` foi alterado como dito previamente e as funções da interface foram adicionadas para permitir que uma aplicação libere o processador, realize ou anule uma reserva de tempo.

4.2 Modificações feitas no mplayer

As modificações realizadas no `mplayer` [19] se concentram no arquivo principal `main.c` e são as seguintes:

- Um flag foi adicionado para permitir que o usuário especifique um período e uma reserva de tempo dentro deste período. Algumas chamadas para a função da interface `set_reservation` foram adicionadas antes e depois do loop principal para inicializar e anular a reserva, respectivamente.

- Na função `ExecuteDisplay`, foi adicionada uma chamada à função `edf_sched_yield()` após o tratamento e exibição de cada *stream* de vídeo.

4.3 Resultados experimentais

Os testes efetuados com o `mplayer` e a política de escalonamento foram realizados em um laptop Toshiba Satellite 2410 Pentium IV 1.8 GHz com 512Mb de memória principal utilizando um trailer do filme Matrix Reloaded. Este vídeo contém 1531 quadros e dura em média 1 minuto e meio. Em segundo plano, foram executados entre 0 e 6 processos realizando a compilação do núcleo Bossa/Linux (através do comando `make bzImage`).

A seguir apresentamos o desempenho da aplicação `mplayer` original executando sob o escalonador Linux e a execução do `mplayer` modificado sob o controle do escalonador EDF/Linux. A versão modificada do `mplayer` efetua uma reserva de tempo fixa de 20 μs a cada 30 μs. Como ilustra o gráfico da Figura 3, a partir de quatro processos de compilação em segundo plano o `mplayer` original apresenta uma degradação progressiva da taxa de quadros exibidos enquanto que o `mplayer` modificado mantém a taxa quadros exibidos no patamar requerido pela aplicação.

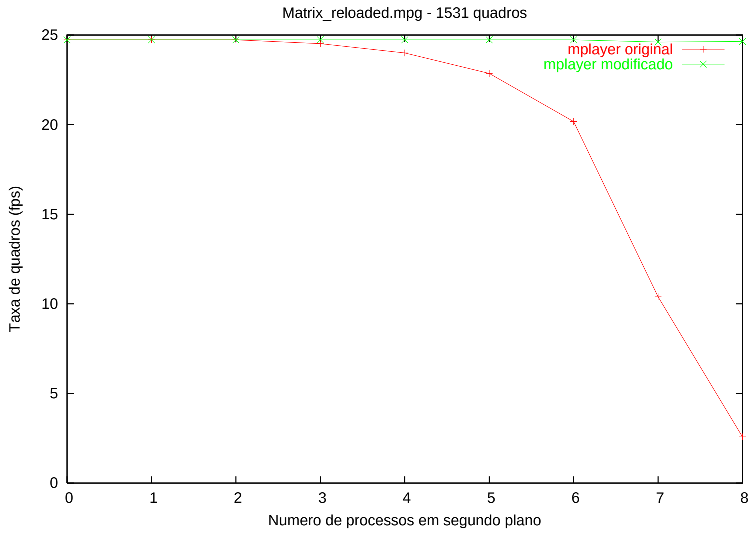

Figura 3: Taxa de quadros média pela aplicação `mplayer`.

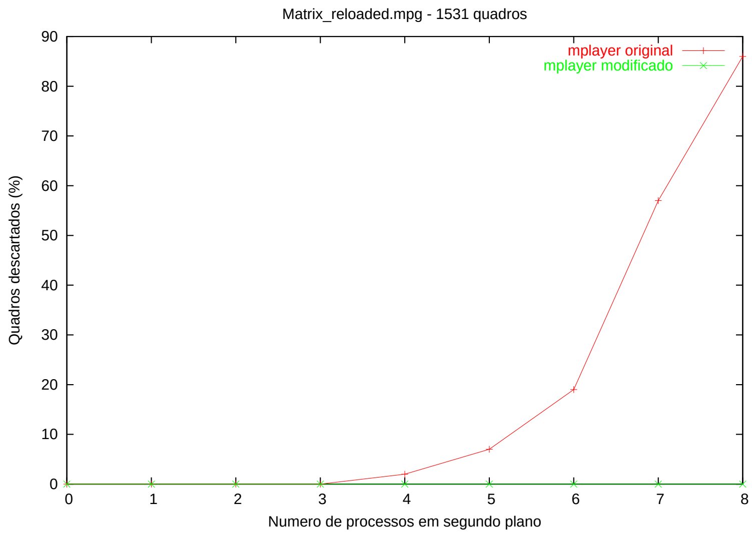

Figura 4: Percentual de quadros descartados da aplicação `mplayer`.

Através do gráfico da Figura 4 verifica-se que o desempenho do `mplayer` modificado é bastante superior ao `mplayer` original em relação ao percentual de quadros descartados. Ainda que o `mplayer` modificado efetue um descarte de quadros a medida que o número de processos em segundo plano aumenta, esse percentual se manteve abaixo de 1 % fazendo com que a curva correspondente se confunda com o eixo das abscissas na Figura 4. Em termos qualitativos, considerando o `mplayer` original nota-se que a partir de cinco processos em segundo plano a sincronização entre áudio e vídeo é extremamente prejudicada e perde-se quase totalmente todos os quadros com oito processos em execução.

5 Trabalhos relacionados

A especialização de serviços do sistema operacional para melhoria de aplicações multimídia tem sido empregada em diversos contextos, tais como, sistemas operacionais extensíveis, *middleware* dedicados e infraestruturas para desenvolvimento de escalonadores de processos. Entretanto, grande parte destas abordagens são específicas a uma arquitetura ou sistema operacional o que as torna dificilmente reutilizáveis na implementação de novas políticas de gestão de recursos.

Os sistemas operacionais ditos *extensíveis*, tais como Exokernel [14] e SPIN [6], permitem aos programadores de sistema estender ou substituir as funcionalidades fornecidas pelo núcleo a fim de melhorar o desempenho das aplicações. Embora esses sistemas tenham demonstrado na prática como melhorar o desempenho de certas aplicações multimídia, sua utilização continua restrita a experts em sistemas operacionais. A programação de extensões em SPIN é feita por meio da linguagem Modula-2 que, apesar de facilitar o desenvolvimento e a verificação de certas politícas do núcleo, não fornece abstrações específicas no tocante ao escalonamento de processos. Além disso, existem restrições quanto à substituição do escalonador do núcleo.

RTLinux [1] é uma distribuição Linux para tempo-real que permite a introdução de novas políticas de escalonamento. O RTLinux possui um modelo de programação específico para os processos de tempo-real, chamados de *Real Time Tasks*. QLinux [32] é outra adaptação do núcleo do Linux que visa garantir qualidade de serviço para diversas classes de aplicações. O QLinux disponibiliza um escalonador hierárquico (baseado no algoritmo *Hierarchical Start Time Queueing* [13]) permitindo classes de processos com diferentes níveis de qualidade de serviço. Assim como na implementação direta no núcleo de distribuições Linux padrão (Redhat, Debian, Mandrake), a ausência de abstrações de mais alto nível no RTLinux e no QLinux requer um conhecimento aprofundado da operação do núcleo para o desenvolvimento de novas políticas de escalonamento.

TAO [27] é uma implementação do padrão CORBA que fornece um modelo de QoS para aplicações de tempo-real através de uma implementação e uma interface específicas. A implementação disponibiliza políticas de escalonamento adaptadas para sistemas de tempo-real como *Rate Monotonic*, *Least Laxity First* e EDF [12]. Ainda é possivel efetuar uma análise dos requisitos das aplicações para definir o escalonamento de maneira estática (*off-line*), o que é essencial para sistemas de tempo-real crítico e que atualmente não é suportado pela plataforma Bossa. Entretanto, é difícil precisar se os mecanismos disponibilizados por TAO são facilmente modificáveis para a integração de novas políticas adaptadas às aplicações multimídia a exemplo da estratégia baseada na noção de progresso da aplicação.

A política de escalonamento baseada na noção de progresso da aplicação `mpeg_play`, apresentada na seção 3, foi inspirada pelo trabalho desenvolvido por Steere *et al.* [31]. No entanto, estas estratégias diferem quanto à definição das métricas de progresso. Ao passo que utilizamos a taxa de quadros atual da aplicação, a política empregada por Steere *et al.* adapta a prioridade dos processos que compõem a aplicação de vídeo (estruturada como um *pipeline* de processos executando tarefas distintas) em função da taxa de preenchimento dos *buffers* de entrada e saída de cada processo. Por exemplo, um processo tendo um *buffer* de entrada cheio e um *buffer* de saída vazio recebe um aumento de prioridade. Entretanto, grande parte do código é escrito diretamente em C no núcleo do Linux, o que dificulta a compreensão e a evolução desse código, ao passo que utilizando a linguagem Bossa, as modificações requeridas no escalonador podem ser facilmente localizadas e não requerem um conhecimento específico do núcleo.

6 Conclusões e trabalhos futuros

O bom funcionamento de aplicações multimídia requer uma gestão de recursos adequada e eficiente por parte do sistema operacional. Para atingir tal objetivo, as aplicações devem poder informar seus requisitos de QoS ao sistema operacional que, por sua vez, deve implementar políticas que atendam às necessidades das aplicações. Este artigo descreveu uma abordagem de desenvolvimento que permite a especialização de escalonadores de processos que juntamente com modificações na aplicação permitem melhorar o desempenho de aplicações multimídia.

Os resultados experimentais obtidos com duas aplicações de vídeo demonstram uma melhora significativa na qualidade de serviço das aplicações quando comparadas às aplicações originais sob a gestão do escalonador original do Linux. Além disso, as modificações efetuadas nas aplicações e nas políticas de escalonamento são simples de serem implementadas e estão ao alcance de programadores convencionais pois não requerem um conhecimento aprofundado do sistema operacional ou do sistema de execução.

A extensão natural deste trabalho reside no desenvolvimento de novas políticas de escalonamento em outros contextos relacionados à multimídia, em particular, políticas voltadas à aplicações de multimídia distribuída, tais como *streaming* de vídeo em tempo real. Neste contexto, seria apropriado o uso de políticas de escalonamento específicas de acordo com as necessidades e a funcionalidade (cliente ou servidor) de cada nodo envolvido na aplicação. Outra possibilidade consiste no desenvolvimento de escalonadores específicos para a gestão de roteadores no contexto de redes ativas, nas quais pode-se instalar programas nos roteadores para, por exemplo, realizar um processamento específico nos pacotes tal como a compressão de dados ou descarte de pacotes. Neste cenário, o escalonador pode limitar o tempo de execução dos programas instalados no roteador [24] ou ainda controlar a importância de cada processo em função do preenchimento dos *buffers* utilizados pelo roteador.

Disponibilidade

A distribuição do núcleo Bossa/Linux, o compilador/verificador e exemplos de políticas de escalonamento encontram-se publicamente disponíveis em
`http://www.emn.fr/x-info/bossa`

Referências

[1] M. Barabanov. A Linux-based realtime operating system. Master's thesis, New Mexico Institute of Mining and Technology, June 1997.

[2] L. P. Barreto, R. Douence, G. Muller, and M. Südholt. Programming OS schedulers with domain-specific languages and aspects: New approaches for OS kernel engineering. In *Proceedings of the 1st Workshop on Aspects, Components, and Patterns for Infrastructure Software*, pages 1–6, April 2002.

[3] L. P. Barreto and G. Muller. Bossa: a language-based approach to the design of real-time schedulers. In *Proceedings of the 10th International Conference on Real-Time Systems (RTS'2002)*, pages 19–31, Paris, France, March 2002.

[4] M. Beck, H. Bohme, M. Dzladzka, U. Kunitz, R. Magnus, and D. Verworner. *Linux Kernel Internals*. Addison-Wesley, Reading, MA, USA, 1998.

[5] Berkeley MPEG Player web site. http://bmrc.berkeley.edu/.

[6] B. N. Bershad, S. Savage, P. Pardyak, E. G. Sirer, M. Fiuczynski, D. Becker, S. Eggers, and C. Chambers. Extensibility, safety and performance in the SPIN operating system. In *Proceedings of the 15th ACM Symposium on Operating Systems Principles (SOSP'95)*, pages 267–284, December 1997.

[7] J. Bruno, E. Gabber, B. Ozden, and A. Silberschatz. Move-to-rear list scheduling: a new scheduling algorithm for providing QoS guarantees. In *Proceedings of the ACM Multimedia (MULTIMEDIA'97)*, Seattle, WA, USA, November 1997.

[8] H. Chu and K. Nahrstedt. CPU service classes for multimedia applications. In *Proceedings of the IEEE International Conference on Multimedia Computing and Systems (ICMCS'99)*, Florence, Italy, June 1999.

[9] J. Douceur and W. Bolosky. Progress-based regulation of low-importance processes. In SOSP'99 [30], pages 247–258.

[10] K. Duda and D. Cheriton. Borrowed-virtual-time (BVT) scheduling: supporting latency-sensitive threads in a general-purpose scheduler. In SOSP'99 [30], pages 261–276.

[11] B. Ford and S. Susarla. CPU inheritance scheduling. In OSDI'96 [21], pages 91–105.

[12] C. Gill, L. Levine, and D. Schmidt. The design and performance of a real-time corba scheduling service. *The International Journal of Time-Critical Computing Systems*, 20, March 2001.

[13] P. Goyal, X. Guo, and H. M. Vin. A hierarchical CPU scheduler for multimedia operating systems. In OSDI'96 [21], pages 107–121.

[14] M. F. Kaashoek, D. R. Engler, G. R. Ganger, H. Briceño, R. Hunt, D. Mazières, T. Pinckney, R. Grimm, J. Jannotti, and K. Mackenzie. Application performance and flexibility on exokernel systems. In SOSP'97 [29], pages 52–65.

[15] Fabio Kon, Roy H. Campbell, M. Dennis Mickunas, Klara Nahrstedt, and Francisco J. Ballesteros. 2K: A Distributed Operating System for Dynamic Heterogeneous Environments. In *Proceedings of the 9th IEEE International Symposium on High Performance Distributed Computing (HPDC'9)*, pages 201–208, Pittsburgh, August 2000.

[16] J. Lawall, G. Muller, and L. P. Barreto. Capturing OS expertise in a modular type system: the Bossa experience. In *Proceedings of the ACM SIGOPS European Workshop 2002 (EW'2002)*, pages 54–61, Saint-Emillion, France, September 2002.

[17] Ian M. Leslie, Derek McAuley, Richard Black, Timothy Roscoe, Paul T. Barham, David Evers, Robin Fairbairns, and Eoin Hyden. The design and implementation of an operating system to support distributed multimedia applications. *IEEE Journal of Selected Areas in Communications*, 14(7):1280–1297, 1996.

[18] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 20(1):46–61, January 1973.

[19] MPlayer web site. http://www.mplayerhq.hu/.

[20] J. Nieh and M. S. Lam. The design, implementation and evaluation of SMART: A scheduler for multimedia applications. In SOSP'97 [29], pages 184–197.

[21] *Proceedings of the 2nd USENIX Symposium on Operating Systems Design and Implementation (OSDI'96)*, Berkeley, CA, USA, October 1996.

[22] D. Petrou, J. W. Milford, and G. A. Gibson. Implementing lottery scheduling: Matching the specializations in traditional schedulers. In *Proceedings of the USENIX Annual Technical Conference (USENIX'99)*, pages 1–14, Berkeley, CA, USA, June 1999.

[23] C. Poellabauer, K. Schwan, and R. West. Coordinated CPU and event scheduling for distributed multimedia applications. In *Proceedings of the 9th International ACM Conference on Multimedia (MULTIMEDIA'2001)*, pages 231–240, September 2001.

[24] X. Qie, A. Bavier, L. Peterson, and S. Karlin. Scheduling Computations on a Software-Based Router. In *Proceedings of the ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS'2001)*, June 2001.

[25] J. Regehr and J. A. Stankovic. Augmented CPU reservations: towards predictable execution on general-purpose operating systems. In RTAS'2001 [26].

[26] *Proceedings of the 7th Real-Time Technology and Applications Symposium (RTAS'2001)*, Taipei, Taiwan, May 2001.

[27] Douglas C. Schmidt, Aniruddha Gokhale, Timothy H. Harrison, and Guru Parulkar. A high-performance endsystem architecture for real-time CORBA. *IEEE Communications Magazine*, 14(2), 1997.

[28] C. Small. *Building an Extensible Operating System*. PhD thesis, Harvard University, Division of Engineering and Applied Sciences, October 1998.

[29] *Proceedings of the 16th ACM Symposium on Operating Systems Principles (SOSP'97)*, October 1997.

[30] *Proceedings of the 17th ACM Symposium on Operating Systems Principles (SOSP'99)*, December 1999.

[31] D. C. Steere, A. Goel, J. Gruenberg, D. McNamee, C. Pu, and J. Walpole. A feedback-driven proportion allocator for real-rate scheduling. In *Proceedings of the 3rd Symposium on Operating Systems Design and Implementation (OSDI'99)*, pages 145–158, February 1999.

[32] V. Sundaram, P. Shenoy, A. Chandra, J. Sahni, P. Goyal, and H. Vin. Application performance in the QLinux multimedia operating system. In *Proceedings of the 8th International ACM Conference on Multimedia (MULTIMEDIA'2000)*, pages 127–136, October 2000.

[33] C. A. Waldspurger and W. E. Weihl. Lottery scheduling: Flexible proportional-share resource management. In *Proceedings of the 1st USENIX Symposium on Operating Systems Design and Implementation (OSDI'94)*, pages 1–11, Berkeley, CA, USA, November 1994.

[34] D. K. Y. Yau and S. S. Lam. Adaptive rate-controlled scheduling for multimedia applications. *IEEE/ACM Transactions on Networking*, 5(4):475–488, August 1997.

[35] W. Yuan, K. Nahrstedt, and K. Kim. R-EDF: A reservation-based EDF scheduling algorithm for multiple multimedia task classes. In RTAS'2001 [26].