UNIVERSIDADE FEDERAL DE SANTA CATARINA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Cristiano Roberto Cervi

ESTUDO DO GERENCIAMENTO DE QOS AO NÍVEL DE SISTEMA OPERACIONAL PARA APLICAÇÕES MULTIMÍDIA DISTRIBUÍDAS

Dissertação submetida à Universidade Federal de Santa Catarina como parte dos requisitos para a obtenção do grau de Mestre em Ciência da Computação

Prof. Dr. Luis Fernando Friedrich
Orientador

Florianópolis, março de 2003

ESTUDO DO GERENCIAMENTO DE QOS AO NÍVEL DE SISTEMA OPERACIONAL PARA APLICAÇÕES MULTIMÍDIA DISTRIBUÍDAS

Cristiano Roberto Cervi

Esta Dissertação foi julgada adequada para a obtenção do título de Mestre em Ciência da Computação na Área de Concentração Sistemas de Computação e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação.

_____

Fernando Alvaro Ostuni Gauthier, Dr
Coordenador do Curso

Banca Examinadora

_____

Luis Fernando Friedrich, Dr (orientador)

_____

José Mazzucco Júnior, Dr

_____

Roberto Willrich, Dr

Dedico este trabalho a meu orientador, aos meus amigos e a minha família.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

API	- Application Programming Interface
ATM	- Asynchronous Transfer Mode
CD	- Compact Disc
CPU	- Central Processor Unit
C-SCAN	- Circular SCAN
DAT	- Digital Audio Tape
DCT	- Discrete Cosine Transform
DPCM	- Differential Pulse Code Modulation
DWT	- Discrete Wavelet Transform
FCFS	- First–Come-First-Served
FDDI	- Fiber Distributed Data Interface
FIFO	- First-In-First-Out
FTP	- File Transfer Protocol
IETF	- Internet Engineering Task Force
I/O	- Input/Output
IP	- Internet Protocol
ISO	- International Standard Organization
ITU	- International Telecommunication Union
JPEG	- Joint Photographic Experts Group
LRU	- Least Recently Used
MPEG	- Motion Picture Experts Group
OPT	- Optimal Replacement
OSI	- Open System Interconection
PCB	- Process Control Block
QoS	- Quality of Service
RDSI-FL	- Rede Digital de Serviços Integrados – Faixa Larga
SMTP	- Simple Mail Transfer Protocol
SO	- Sistema Operacional
SSTF	- Shortest-Seek-Time-First
RSVP	- Resource Reservation Protocol
RT-VBR	- Real-Time Variable Bit Rate
TCP	- Transmission Control Protocol
TCP/IP	- Transmission Control Protocol / Internet Protocol
UDP	- User Datagram Protocol

RESUMO

Qualidade de serviço (QoS) é uma especificação qualitativa e quantitativa dos requisitos de uma aplicação que um sistema multimídia deve satisfazer a fim de obter a qualidade desejada [Lu, 96]. Para garantir a qualidade de serviço necessária o sistema deve fazer uma verificação dos recursos disponíveis e reservar uma certa quantia desses recursos em função de requisitos que deverão ser passados como parâmetros ao sistema. Se houver disponibilidade o sistema faz a reserva, caso contrário deve informar que não há recursos disponíveis e terminar o processo ou ainda pode propor uma renegociação dos requisitos para que a aplicação seja executada mesmo sem os requisitos solicitados na primeira negociação.

Este trabalho faz um estudo sobre o gerenciamento de QoS ao nível de sistema operacional onde mostra o conjunto de requisitos necessários que um sistema operacional deve conter para o fornecimento da garantia de qualidade de serviço para aplicações multimídia distribuídas dando enfoque na reserva de recursos, gerenciamento de recursos e escalonamento. Ainda faz um estudo das plataformas Windows, através do Windows 2000 e Unix, através do Linux, para ver se algum destes sistemas possui esse conjunto de requisitos.

Palavras-chave: Multimídia, Sistema Operacional, Qualidade de Serviço, QoS.

ABSTRACT

Quality of Service (QoS) is a qualitative and quantitative specification of the requirements of an application that a system multimedia must satisfy in order to get the desired quality [Lu, 96]. To guarantee the necessary quality of service the system must make a verification of the available resources and reserve a certain amount of these resources in function of requirements that will have to be passed as parameters to the system. If the system will have availability makes the reserve, contrary case must inform that it does not have available resources and to finish the process or still can consider a renegotiation of the requirements so that the application is executed same without the requested requirements in the first negotiation.

This work makes a study on the management of QoS to the level of operating system where it shows the set of necessary requirements that an operational system must contain for the supply of the assurance quality of service for applications multimedia distributed giving approach in the reserve of resources, management of resources and scheduling. Still it makes a study of the Windows platforms, through the Windows 2000 and Unix, through the Linux, to see if some of these systems possesss this set of requirements.

Keywords: Multimedia, Operating System, Quality of Service, QoS.

1. Introdução

A comunicação multimídia promete efetivamente mudar a forma como o ser humano se relaciona, diminuindo as distâncias intercontinentais, fazendo com que fatos que acontecem do outro lado do planeta afetem o cotidiano de uma população distante.

Alguns fatores fundamentais para a concretização da transmissão multimídia são, entre outros, o aumento da capacidade de processamento, permitindo a execução de aplicações gráficas, o amadurecimento dos sistemas distribuídos, com uma melhoria das redes de computadores e a popularização da Internet, e o desenvolvimento de novas técnicas de compressão digital.

As principais aplicações, onde se destacam o áudio e a videoconferência, os serviços sob demanda, a telemedicina, aprendizado remoto, trabalho cooperativo, entre outros, se caracterizam fundamentalmente pelo alto volume de dados que envolvem e pelas dependências temporais nas apresentações. Tais recursos são denominados mídias contínuas, e precisam ser consideradas com mais atenção para a implementação de um sistema.

As mídias de áudio e vídeo apresentam algumas características particulares que devem ser levadas em consideração na concepção e na utilização de um sistema multimídia distribuído, pois afetam diretamente a qualidade final das informações que são apresentadas ao usuário.

As principais características se referem às restrições temporais e a tolerância a perdas. A primeira restrição temporal se refere ao atraso fim-a-fim onde deve ser bem controlado, caso contrário a comunicação pode ficar inviável. A segunda questão é relativa a variação do atraso fim-a-fim.

A exibição de mídias como áudio e vídeo deve respeitar uma determinada velocidade de apresentação, que é natural para a compreensão e percepção do usuário. A origem da mídia transmite os dados na taxa natural, e o sistema de interconexão, como a rede, deve respeitar essa taxa. Se a rede, por qualquer motivo, atrasar algumas

informações de voz, ao chegar no receptor não adianta apresentá-las mais rapidamente. Estes dados acumulados podem ser descartados, pois já passou seu tempo de apresentação, devendo-se retomar a exibição dos dados que estão no tempo certo.

Existem alguns métodos que foram desenvolvidos para amenizar esse problema. Um deles é o armazenamento de uma determinada quantidade de informação antes de sua apresentação no dispositivo de exibição. Este método, por sua vez, introduz um atraso, que será tanto maior quanto maior a quantidade de informação acumulada. Desta forma, o tamanho da memória é um compromisso entre o atraso e a flexibilidade de poder proporcionar uma apresentação contínua, sem interrupções ou perdas. Devido a essa característica, mídias como áudio e vídeo são denominadas mídias contínuas.

A última questão temporal é o sincronismo. Por exemplo, em transmissões de vídeo com áudio associado, é fundamental que essas duas mídias sejam apresentadas em sincronia. Um dos casos mais tradicionais dessa restrição é denominado sincronismo labial, onde a fala de uma pessoa deve acompanhar os movimentos de seu lábio. A percepção desse fenômeno depende de vários fatores, mas em condições mais comuns de telejornalismo, por exemplo, o tempo máximo de diferença de apresentação voz-vídeo está na ordem de poucos décimos de segundo.

No que se refere a perdas, as mídias contínuas são menos exigentes. Tendo em vista as propriedades cognitivas do sistema visual e auditivo humano, a perda de pequenos trechos de informações, face a rapidez com que estas são apresentadas, pode ser tolerada. Contudo, sistemas de compressão dedicados para as mídias contínuas, tal como o MPEG, procuram codificar somente as informações mais relevantes para o entendimento humano, conseguindo elevadas reduções de volume do conjunto original de dados, com um pouco de degradação da qualidade. Isso, por sua vez, também implica que alguma perda de informações codificadas pode não ser desprezível, já que se procura transmitir somente os dados de maior importância.

A qualidade final das mídias que serão apresentadas ao usuário é composta pelo conjunto dessas características somadas com as especificações do sistema, como tamanho da janela, quantidade de cores (preto e branco, 16, 256 cores, todas as cores existentes), quantidade de quadros de vídeo apresentados por segundo (10,15,30, 60),

qualidade do som (mono, estéreo, qualidade de CD) etc. Conhecendo o que se quer implementar, com todas suas restrições, o segundo passo é saber qual a infra-estrutura que será utilizada, composta basicamente pelos sistemas computacionais e de rede.

O grande avanço da tecnologia dos processadores têm garantido um aumento de desempenho e queda de custos, de modo a oferecer, a custos razoáveis, sistemas computacionais de uso geral capazes de lidar com as mídias contínuas apresentando qualidade aceitável para uma videoconferência ou a exibição de um vídeo comprimido.

Já no âmbito das redes de computadores, que interligam esses sistemas computacionais, o quadro não é o mesmo. Apesar do surgimento de novas tecnologias de rede, que procuram oferecer recursos especiais para a garantia da banda-passante e do atraso fim-a-fim, como a ATM, o emprego de tais tecnologias, atualmente, ainda está mais restrita ao *backbone* de redes corporativas ou a enlaces de longa distância oferecidos pela empresas de telecomunicações. Sua aplicação fim-a-fim ainda é discutível, tendo em vista ainda elevado preço da tecnologia e a carência de aplicativos que explorem todos seus recursos de garantias de qualidade de serviço.

Por outro lado, a Internet cresce muito rápido, oferecendo uma conectividade nunca vista antes, interligando todas as partes do mundo, a preços acessíveis ao usuário final. Porém, sua infra-estrutura, baseada fundamentalmente no protocolo IP, ou na pilha de protocolos TCP/UDP-IP, ainda não oferece suporte adequado ao transporte das mídias contínuas e a garantia de qualidade de serviço em apresentações multimídia.

A noção de qualidade de serviço ou QoS (*Quality of Service*) surgiu para descrever certas características técnicas de transmissão de dados em redes de computadores e apesar de seu conceito não ser novo, apenas recentemente ele passou a ser mais difundido, especialmente com o surgimento das redes de alta velocidade, como RDSI-FL (Rede Digital de Serviços Integrados – Faixa Larga) e ATM (*Asynchronous Transfer Mode* ou Modo de Transmissão Assíncrona).

Também com o surgimento das redes de alta velocidade, foi possível se pensar em apresentar dados multimídia de forma distribuída, já que estes requerem muitos requisitos, tanto de software como de hardware.

As aplicações multimídias são consideradas aplicações de tempo real, ou seja, são essencialmente dependentes do tempo, pois exigem uma resposta dentro de um certo limite de tempo.

Antes de uma aplicação multimídia iniciar sua execução é necessário que certas garantias sejam cumpridas para que ela possa ser executada sem falhas ou interrupções. Essa garantia é dada por um contrato de qualidade de serviço acertado entre a aplicação e o sistema, levando-se em consideração alguns requisitos como tempo de apresentação, tempo de CPU (*Central Processing Unit*), largura de banda de transmissão e suporte do sistema operacional (SO).

Para garantir a qualidade de serviço necessária o sistema deve fazer uma verificação dos recursos disponíveis e reservar uma certa quantia desses recursos em função de requisitos que deverão ser passados como parâmetros ao sistema. Se houver disponibilidade o sistema faz a reserva, caso contrário deve informar que não há recursos disponíveis e terminar o processo ou ainda pode propor uma renegociação dos requisitos para que a aplicação seja executada mesmo sem os requisitos solicitados na primeira negociação.

Para este trabalho utilizaremos a definição de [Lu, 96] onde caracteriza qualidade de serviço como sendo uma especificação qualitativa e quantitativa dos requisitos de uma aplicação que um sistema multimídia deve satisfazer a fim de obter a qualidade desejada.

Este trabalho faz um estudo sobre o gerenciamento de QoS ao nível de sistema operacional onde mostra o conjunto de requisitos necessários que um sistema operacional deve conter para o fornecimento da garantia de qualidade de serviço para aplicações multimídia distribuídas dando enfoque na reserva de recursos, gerenciamento de recursos e escalonamento. Ainda faz um estudo das plataformas Windows, através do Windows 2000 e Unix, através do Linux, para ver se algum destes sistemas possui esse conjunto de requisitos.

Essa dissertação faz um estudo sobre o gerenciamento de QoS ao nível de sistema operacional e mostrar o conjunto de requisitos necessários que um sistema

operacional deve conter para o fornecimento da garantia de qualidade de serviço para aplicações multimídia distribuídas dando enfoque na reserva de recursos, gerenciamento de recursos e escalonamento. Ainda faz um estudo das plataformas Windows, através do Windows 2000 e Unix, através do Linux, para ver se algum destes sistemas possui mecanismos diferenciados de gerenciamento de QoS.

Esta dissertação está organizada na forma que segue:

Inicialmente, alguns conceitos básicos da área de multimídia são introduzidos no capítulo 2;

O capítulo 3 apresenta noções de qualidade de serviço com uma atenção especial ao protocolo de reserva de recursos RSVP;

No capítulo 4 são abordados alguns conceitos acerca de sistemas operacionais;

No capítulo 5 são apresentados os requisitos que um sistema operacional deve ter a fim de garantir qos para aplicações multimídia distribuídas;

No capítulo 6 é apresentado um estudo abordando a alocação de memória, gerência de memória e escalonamento no Linux e no Windows 2000;

No capítulo 7 é apresenta a conclusão.

2. Multimídia

O objetivo deste capítulo é introduzir alguns conceitos sobre multimídia, suas aplicações, a necessidade de compressão e ainda identificar alguns requisitos de rede e hardware para transmissão de áudio e vídeo em aplicações multimídia distribuídas.

2.1. Definição de Multimídia

Segundo [Fluckiger, 95] multimídia é o campo interessado na integração controlada por computador de textos, gráficos, imagens, vídeos, animações, sons e qualquer outro meio onde todo tipo de informação pode ser representada, armazenada, transmitida e processada digitalmente.

2.2. Classificação dos Tipos de Mídia

Os vários tipos de mídia podem ser identificados quanto ao comportamento temporal de apresentação de informação multimídia. Neste caso, as mídias podem ser classificadas em:

- *Mídias discretas:* também chamadas de estáticas, pois não dependem do tempo, são mídias com dimensões unicamente espaciais, como textos, imagens e gráficos;

- *Mídias contínuas:* também chamadas de dinâmicas ou isócronas, são mídias com dimensões temporais que dependem do tempo em que são apresentadas, como por exemplo sons, vídeos e animações.

2.3. Sistemas Multimídia

Vários autores definem sistemas multimídia como sistemas suportando a apresentação de ao menos uma mídia discreta e uma mídia contínua ao mesmo tempo, ambas representadas na forma digital.

2.4. Classificação dos Sistemas Multimídia

Os sistemas multimídia estão divididos em dois grupos: um concentra seus esforços em estações de trabalho multimídia *standalone*, sistemas de software associados e ferramentas e o outro grupo combina computação multimídia com sistemas distribuídos oferecendo maiores potencialidades.

2.4.1. Sistemas Multimídia *Standalone*

Aplicações multimídia em sistemas *standalone* utilizam apenas recursos locais para serem executadas. Todas as informações necessárias devem estar armazenadas localmente, pois não possuem capacidade de armazenamento remoto.

O sistema local deve fornecer toda capacidade de processamento necessário e ainda estar equipado com dispositivos de captura e apresentação. Podemos citar como uma aplicação *standalone*, alunos tendo aula e as apresentações estarem sendo executadas cada uma em um computador.

2.4.2. Sistemas Multimídia Distribuídos

Ao contrário das aplicações *standalone*, as aplicações multimídia distribuídas necessitam de um ambiente distribuído, ou seja, utilizam uma rede de comunicação para serem executadas, onde os sistemas locais acessam os servidores remotamente e podem compartilhar e armazenar informações.

Justifica-se esse tipo de utilização, pois atualmente busca-se cada vez mais um serviço para fornecer informações multimídia à distância, como por exemplo, a videoconferência.

2.5. Tipo de Transferência de Dados Multimídia

[Fluckiger, 95] classifica as aplicações multimídia distribuídas em duas classes:

- *Aplicações pessoa-a-pessoa:* onde o principal objetivo é aumentar a comunicação entre humanos, como através da Videofonia ou da Videoconferência;

- *Aplicações pessoa-a-sistema:* onde uma pessoa ou um grupo de pessoas se comunica com sistemas remotos para acessar, receber ou interagir com informações multimídia.

2.5.1. Aplicações Multimídia pessoa-a-pessoa

Podem ser divididas em aplicações síncronas e assíncronas.

- *Aplicações síncronas:* também chamadas de tempo real são aquelas onde a informação é gerada em tempo real pelo emissor e apresentada no receptor assim que estiver disponível. Dessa categoria fazem parte as aplicações:
    - *interpessoais*: onde apenas dois indivíduos estão envolvidos, como na videofonia;
    - *de distribuição*: onde as informações como áudio e vídeo são transmitidos ao vivo de uma fonte para vários destinos;
    - *teleconferência de grupo*: comunicação conversacional bidirecional entre dois ou mais grupo de pessoas.
- *Aplicações assíncronas:* são aquelas onde o instante de leitura da apresentação é diferente do tempo em que ela foi transmitida pelo emissor, ficando a cargo do usuário final a decisão do momento em que a informação enviada será executada. Como exemplo temos:
    - *e-mail multimídia*: semelhante ao correio eletrônico tradicional, contém texto formatado, áudio e vídeo;
    - *conferência multimídia assíncrona*: pessoas mantém e seguem conversações não sincronizadas através de lista de tópicos onde a mensagem trocada é multimídia.

2.5.2. Aplicações Multimídia Pessoa-a-Sistema

Podem ser divididas em aplicações interativas e de distribuição.

- *Aplicações interativas:* onde o iniciador da comunicação é o usuário final, que a qualquer momento pode pedir a informação a um servidor. Dessa categoria fazem parte as aplicações:
    - *orientada a busca*: onde o objetivo é localizar, acessar e apresentar informações multimídia;
    - *orientada a transação*: onde a entrada do usuário é processada a outros propósitos que controlar a apresentação da informação.
- *Aplicações de distribuição:* são aquelas onde a distribuição da informação multimídia é iniciada no servidor. Como exemplo temos:
    - *distribuição para grupos fechados*: onde somente usuários selecionados recebem a informação;
    - *distribuição para grupos abertos*: não existe nenhuma restrição e a única condição para os receptores potenciais se juntarem ao grupo aberto é estarem conectados à rede de distribuição e ainda terem softwares e equipamentos apropriados.

2.6. Requisitos e Características dos Dados Multimídia

Segundo [Lu, 96], dados multimídia tem características e requisitos diferentes de dados alfanuméricos:

- Ddados multimídia devem satisfazer requisitos de tempo real, pois em muitas aplicações, são transmitidos, processados e apresentados com uma taxa fixa;
- Dados multimídia são muito grandes para serem transmitidos com as atuais redes de comunicação que utilizamos, por isso devem ser utilizadas redes de alta velocidade e ainda compactar os dados no servidor antes da transmissão e descompactar na máquina cliente;

- Como as aplicações multimídia usam simultaneamente vários tipos de mídia, as relações temporais e espaciais entre essas mídias devem ser mantidas.

2.7. Requisitos de Rede

Os principais requisitos que a transmissão de dados multimídia impõem às redes de comunicação são expressos em termos de armazenamento, largura de banda de rede e características de desempenho como vazão, confiabilidade, atraso e variação de atraso.

Os sistemas multimídia distribuídos utilizam, por razões econômicas, um tipo de rede por comutação de pacotes ao invés de circuitos integrados [Lu, 96].

2.7.1. Armazenamento e Largura de Banda de Rede

Para medirmos o requisito de armazenamento utilizamos bytes ou Mbytes e a largura de banda é medida como taxa de bits em bits/s ou Mbits/s. A unidade de medida, respectivamente é byte e bit.

Segundo [Willrich, 00], o requisito de armazenamento para imagens pode ser calculado a partir do número de *pixels* (H) em cada linha, o número de linhas (V) na imagem e o número de bits por *pixel* (P), da seguinte maneira: **Requisito de Armazenamento = HVP/8**. Em uma imagem com 480 linhas, 600 *pixels* em cada linha e 24 bits por linha, precisa 864 Kbytes para representar a imagem.

Para calcularmos a largura de banda para transmitir a imagem utilizamos o requisito de armazenamento. Se precisarmos transmitir 864 Kbytes em 2 segundos, a largura de banda necessária é 3,456 Mbits/s. Em muitas aplicações, imagens devem ser apresentadas em sincronia com mídias contínuas como áudio. Nesse caso, a transmissão de imagem impõe tempo restrito e requisitos de largura de banda.

A tabela abaixo apresenta os requisitos de largura de banda de áudio e vídeo de diferentes qualidades:

Aplicações	Taxa de Transmissão (Kbits/s)
CD – Áudio	1.411,2
DAT	1.536
Telefone digital	64
Rádio digital, long play DAT	1.024
Vídeo de qualidade de televisão	216.000
Vídeo de qualidade de VHS	54.000
HDTV	864.000

Tabela 1 – Requisitos de largura de banda para áudio e vídeo

A tabela abaixo mostra os requisitos de armazenamento para mídias estáticas e dinâmicas comuns de diferentes durações:

Aplicações	Taxa de Armazenamento (Mbytes)
Livro de 500 páginas	1
100 imagens monocromáticas	7
100 imagens coloridas	100
1h de áudio qualidade de telefone	28,8
1h de áudio CD	635
1h de vídeo qualidade VHS	24,3
1h TV	97000
1h HDTV	389000

Tabela 2 – Requisitos de armazenamento para mídias estáticas e dinâmicas

2.7.2. Requisitos de Vazão

A vazão de uma rede é sua taxa de bits efetiva, ou a sua largura de banda efetiva o que a torna um requisito básico para aplicações multimídia distribuídas. Se não tivermos uma grande vazão, não teremos uma apresentação de qualidade.

Na maioria das redes, seja ela local ou de longa distância, a vazão varia com o tempo. Algumas vezes ela pode mudar rapidamente devido à falhas nos nós da rede ou no congestionamento, quando grandes fluxos de dados são introduzidos.

2.7.3. Taxa de Erro

Este parâmetro pode ser definido de diversas maneiras:

- *Taxa de erro de bits:* é a razão entre o número médio de bits corrompidos ou errados e o número total de bits que são transmitidos;

- *Taxa de erro de pacotes:* é a razão entre o número médio de pacotes corrompidos ou errados e o número total de pacotes transmitidos;

- *Taxa de erro de quadro:* é aplicada a redes ATM, definida como o número de quadros errados pelo total de quadros transmitidos.

Os erros ocorrem mais em redes de comutação de pacotes, quando bits individuais em pacotes são perdidos ou invertidos, pacotes são perdidos no caminho, pacotes são cortados ou atrasados, ou ainda quando pacotes chegam fora de ordem.

Nas redes atuais, as taxas de erros de bits são muito baixas. Em redes orientadas a conexão, por exemplo, o receptor é normalmente capaz de detectar a situação e avisar o emissor que ocorreu algum problema. Já nas redes não orientadas a conexão, os erros são mais difíceis de serem detectados.

2.7.4. Atraso Fim-a-Fim e Variação de Atraso

Em sistemas multimídia distribuídos, o tempo que um bloco de dados leva para ser transmitido do emissor ao receptor é denominado atraso fim-a-fim e é composto pelos seguintes componentes:

- *Atraso de trânsito:* tempo de propagação necessário para enviar um bit de um local para outro;

- *Atraso de transmissão:* tempo necessário para transmitir um bloco de dados fim-a-fim;

- *Atraso de interface:* atraso ocorrido entre o tempo em que o dado está pronto para ser transmitido e o tempo em que a rede está pronta para transmitir o dado.

Nas redes de comutação de pacotes, os dados não chegam ao destino em intervalos fixos como necessário para transmissão de mídias contínuas. Devido a essa variação de atraso os pacotes de áudio e vídeo que chegam no receptor não podem ser apresentados imediatamente, uma vez que a qualidade do som e da imagem ficariam bastante prejudicados.

De acordo com [Willrich, 00], as variações de atraso são muito comuns, pois hoje em dia nas redes por comutação de pacotes os atrasos na transmissão podem ser causados por vários fatores, dentre eles a diferença de tempo de processamento dos pacotes, diferenças de tempo de acesso à rede e diferenças de tempo de enfileiramento.

2.8. Requisitos de Hardware

Para as aplicações multimídias o hardware deve ter um alto poder de processamento e uma alta taxa de transferência de dados, pois áudio e vídeo digital implicam em uma grande quantidade de dados.

Em muitas aplicações são exigidos vários dispositivos de entrada e saída ao mesmo tempo o que torna preferido um tipo de arquitetura paralela.

A arquitetura de hardware deve ser escalável para acomodar novos dispositivos de entrada e saída, bem como novas aplicações.

Um dos principais requisitos de hardware é a capacidade de armazenamento das informações multimídia, já que áudio e vídeo, diferentemente das mídias estáticas, possuem arquivos muito grandes e apesar da utilização das técnicas de compactação dos dados para reduzir os requisitos de armazenamento e também a largura de banda de transferência, o tamanho dos dados ainda continua sendo considerável.

Os discos magnéticos têm as características desejáveis para suportar aplicações multimídia, pois permitem um acesso randômico rápido e tem altas taxas de transferência, mas, em contrapartida, são relativamente caros comparados a outros dispositivos de armazenamento.

Os discos óticos têm mais alta capacidade que discos magnéticos e também permitem o acesso randômico, mas o tempo de acesso é grande e a taxa de transferência é baixa.

As fitas DAT (*Digital Áudio Tape*) têm a maior capacidade de armazenamento, mas não podem ser acessadas randomicamente e a taxa de transferência é baixa.

2.9. Requisitos de Sistema Operacional

O sistema operacional é um dos principais componentes de um sistema multimídia. No entanto, ele deve atender uma série de requisitos para que se possa executar aplicações multimídia com qualidade.

É necessário que possa executar tanto aplicações multimídia como aplicações tradicionais, independentemente se houver solicitação de garantia de QoS. Deve oferecer reserva de recursos e escalonamento de tarefas de acordo com sua disponibilidade.

O gerenciamento de recursos em sistemas operacionais compreende algumas tarefas básicas:

- *Especificação e Requisição de Alocação:* para recursos que são requeridos ao executar tarefas com a qualidade de serviço solicitada;
- *Mecanismos de Escalonamento:* asseguram que um compartilhamento suficiente dos recursos esteja disponível todo o tempo. O tipo de mecanismo depende do tipo de recurso e recursos exclusivos, como CPU e entrada/saída (E/S) de disco devem ser escalonados.

Os requisitos de sistema operacional serão estudados mais detalhadamente no capítulo 5.

2.10. Requisitos de QoS em Redes

Em redes, o conceito de QoS é utilizado para especificar um conjunto de parâmetros de requisitos. Não há um acordo universal deste conjunto, mas parâmetros de requisitos mais comuns como largura de banda (taxa de transferência), limitações de atraso e variações de atraso e ainda sincronização espacial e temporal.

QoS é um contrato negociado entre aplicações e sistemas multimídia (provedores de serviço) e quando uma aplicação necessita uma sessão ela submete um pedido com a QoS requerida para o sistema. O sistema pode aceitar ou propor uma negociação, levando em consideração os seus recursos disponíveis. Quando o sistema

aceita o pedido, um contrato entre o sistema e a aplicação é firmado cabendo ao sistema garantir a QoS solicitada do início ao fim da aplicação.

Existem três formas de garantir o contrato:

- *Garantia Determinística (hard):* onde a QoS deve ser satisfeita completamente;
- *Garantia Estática (soft):* onde o sistema fornece uma garantia com uma certa probabilidade;
- *Melhor Esforço:* onde não existe garantia que a QoS será respeitada.

Os requisitos de QoS serão estudados com mais detalhes no capítulo 3.

2.11. Compressão e Descompressão

Como dados multimídia necessitam de uma grande capacidade de armazenamento, uma grande largura de banda para serem transmitidos e uma grande velocidade dos dispositivos de armazenamento [Furht, 94], as técnicas de compressão e descompressão são fundamentais para que os dados ocupem um espaço aceitável em disco e possam ser transmitidos com uma taxa razoável. Dessa forma entendemos compressão de dados como sendo a codificação de um conjunto de informações onde o código gerado seja menor que a fonte.

Sistemas de compressão exigem dois algoritmos [Tanembaum, 96]: um para a compactação dos dados na origem e outro para a descompactação dos dados no destino. Também chamados de, respectivamente, algoritmos de codificação e decodificação.

Tais algoritmos possuem algumas diferenças, como por exemplo: um filme só será codificado uma única vez, quando for armazenado em um servidor, mas será decodificado várias vezes, sempre que for visto pelo usuário. Isto implica em dizer que o algoritmo de codificação pode ser lento, mas irá necessitar de hardware elevado para ser processado. Já o algoritmo de decodificação, neste caso, precisa ser extremamente rápido e com hardware de baixo custo.

Em outros tipos de aplicações, os requisitos são diferentes e conseqüentemente os algoritmos utilizados também são diferentes. Em uma videoconferência, por exemplo, a codificação e a decodificação devem ser extremamente rápidas, por se tratar de uma aplicação em tempo real, onde à medida que os dados são codificados na origem imediatamente vão sendo transmitidos e decodificados no destino.

2.11.1. Classificação das Técnicas de Compressão

As várias técnicas de compressão existentes podem ser classificadas de diversas maneiras: baseadas no algoritmo de compressão e nos resultados das técnicas de compressão. A seguir serão apresentadas algumas técnicas de compressão quanto ao resultado, já que para este estudo esse tipo de técnica é mais indicado:

- *Compressão com perda:a*s técnicas de compressão com perdas são basicamente utilizadas para transportar som, vídeo e imagem, onde é possível se admitir algum tipo de perda de informação, pois se baseia em propriedades da percepção humana;

- *Compressão sem perdas:* se a informação pode ser exatamente reconstruída após a compressão ela é dita sem perda. Utiliza codificação *Huffman*, codificação aritmética e codificação *Run Length*. Uma das principais áreas onde este tipo de técnica é empregada é no transporte de informações médicas, onde a informação não pode sofrer nenhum tipo de perda

- *Compressão por entropia:* trata de cadeias de bits sem levar em conta o seu significado. É uma técnica sem perda e reversível que pode ser aplicada em todos os tipos de dados. Como exemplos temos a codificação *run-length* e de *Huffman*. A codificação run-length é uma das técnicas mais comuns em multimídia, nela os bits repetidos de uma string são substituídos por um marcador especial, seguido de um símbolo que identifica a operação, seguido do número de vezes que ela ocorreu. Como exemplo, considere a string 654310460000000072111111111111111452. Se **A** for usado como

marcador a string codificada passa a ser 65431046**A08**72**A114**452. Ao invés de transmitir 35 caracteres seriam transmitidos apenas 20;

- *Compressão por Codificação na Origem:* leva em consideração a semântica dos dados onde processa o dado original distinguindo o dado relevante do irrelevante. Depois que os dados irrelevantes forem removidos, os dados relevantes são comprimidos. Como exemplo temos DPCM (*Differential Pulse Code Modulation*), DCT (*Discrete Cosine Transform*) e DWT (*Discrete Wavelet Transform*);

- *Híbrida:* é a combinação das técnicas de compressão sem perdas e codificação na origem. Como exemplo podemos citar os padrões H.261, H.263, JPEG (J*oint Photographic Experts Group*) e MPEG (*Motion Picture Experts Group*). O JPEG é muito utilizado para compressão de imagens estáticas com tons contínuos, como fotografias. Já o MPEG é utilizado para compressão de vídeo, sendo um padrão internacional desde 1993.

2.12. Sincronização

Sincronização multimídia tenta assegurar a ordem temporal desejada entre um conjunto de componentes multimídia em cenário multimídia, o qual denota as semânticas temporais de uma sessão multimídia [Little, 96]. Assim, um esquema de sincronização define os mecanismos usados para obter a sincronização requerida.

O aparecimento temporal correto e desejado de componentes multimídia em uma aplicação possui três significados quando usado em diferentes situações [Lu, 96]:

- *Sincronização intramídia:* quando usado para um fluxo contínuo. Isso significa que as amostras de um áudio e os quadros de um vídeo devem ser apresentados em um intervalo fixo;

- *Sincronização intermídia:* quando usado para descrever os relacionamentos temporais entre componentes multimídia, significa que tais relacionamentos devem ser mantidos. Esse tipo de sincronização é

relacionado com a manutenção das relações temporais entre componentes envolvidos em uma aplicação;

- *Sincronização interação:* quando usados em aplicações interativas. Significa que a resposta correta deveria ser fornecida em um tempo relativamente curto para obter uma boa interação. Esse tipo de sincronização está relacionado com a manutenção de que a correta resposta ocorra em um tempo relativamente curto.

A dependência temporal pode ser implícita e explícita. Para dependência implícita, a relação do tempo é capturada no tempo de aquisição da mídia, como por exemplo, uma sincronização labial, onde a voz deve acompanhar o movimento dos lábios de quem está falando. Na dependência explícita a relação do tempo é criada explicitamente, como por exemplo, em um slide com figuras e anotações de texto.

Na maioria das vezes, a sincronização multimídia pode ser dividida em dois níveis [Wolf, 97]:

- *Especificação temporal:* é o nível mais alto onde mostra as relações temporais entre o conjunto de objetos multimídia abstratos de um cenário de apresentação, onde um objeto abstrato denota uma unidade lógica de dados de mídia, tal como, vídeo ou texto;

- *Sincronização temporal:* é o nível mais baixo, pois traduz a especificação temporal de um cenário apresentacional em uma seqüência de apresentações desejada e realiza a reprodução dos dados dentro de um certo prazo, desconsiderando atrasos da rede, do servidor ou do sistema final.

3. Qualidade de serviço

Como nas aplicações multimídia distribuídas existe a necessidade de assegurar a execução de tais aplicações em tempo real, surgiu a noção de qualidade de serviço ou QoS (*Quality of Service*) que é aplicada como garantia de desempenho em redes de transmissão de dados.

A idéia é que dentre as várias aplicações multimídia existentes nem todas necessitam da mesma garantia de desempenho da rede de transmissão e que estas possam informar os seus requisitos como parâmetros para a apresentação, como largura de banda da rede, atraso, variação de atraso e taxa de perdas.

3.1. Definição de QoS

Atualmente existem muitas definições de QoS. O *International Telecommunication Union* (ITU), refere-se a QoS como um conjunto de requisitos de qualidade de comportamento coletivo de um ou mais objetos. A ATM *Lexicon* define QoS como um termo o qual refere-se a um conjunto de parâmetros de performance ATM que caracteriza o tráfego sobre uma dada conexão virtual e o *Internet Engineering Task Force* trata QoS para ATM como parâmetros para aplicações em tempo real e considera reserva de recursos como meio de oferecer a QoS desejada pelas aplicações, sendo definida através desses parâmetros.

Além dessas definições podemos encontrar várias outras:

- [Wolf, 97] um conjunto de parâmetros que que define as propriedades do fluxo de dados multimídia;
- [Lia, 97] uma descrição quantitativa de quaisquer serviços providos pelo sistema que satisfazem as necessidades da aplicação, sendo esta descrição expressa como um conjunto de pares de parâmetros;

- [Watson, 96] em termos de rede é um conceito pelo qual aplicações podem indicar seus requisitos específicos para a rede antes de iniciar, de fato, a transmissão dos dados;

- [Lu, 96] uma especificação qualitativa e quantitativa dos requisitos de uma aplicação que um sistema multimídia deve satisfazer a fim de obter a qualidade desejada.

Baseado na definição de [Lu, 96], existem dois aspectos para a QoS: aplicações que especificam os requisitos de QoS e sistemas que fornecem as garantias de QoS.

Ao contrário dos serviços tradicionais de transmissão de dados, como *File Transfer Protocol* (FTP) e *Simple Mail Transfer Protocol* (SMTP), em que as variações na transmissão são freqüentemente despercebidas, os vídeos e os áudios são úteis somente se esta variação de atraso estiver dentro de um limite especificado.

Um outro parâmetro de desempenho de rede é o atraso, que significa o tempo levado para transmitir um bloco de dados de um emissor a um receptor.

A noção de QoS foi inicialmente usada em comunicações de dados para caracterizar o desempenho da transmissão dos dados em termos de confiabilidade, atraso e vazão. No modelo de referência OSI (*Open System Interconection*), por exemplo, tem alguns parâmetros de QoS que descrevem a velocidade e a confiabilidade da transmissão, tal como a vazão, o atraso de trânsito, e a taxa de erro e a probabilidade de falha no estabelecimento da conexão.

Os parâmetros de QoS do modelo OSI são especificados na camada de transporte e não têm seus significados diretamente controlados pela aplicação. Estes parâmetros não atendem a todos os requisitos da comunicação multimídia e são apenas usados em nível de transporte, o que nos permite dizer que a qualidade de serviço OSI está incompleta e inconsistente. Nenhum mecanismo é especificado no modelo OSI para garantir a QoS para os requisitos solicitados. Para as comunicações multimídia, a QoS deve ser especificada e garantida fim-a-fim em todos os níveis. Portanto, as aplicações multimídia requerem um novo modelo de QoS.

3.2. Estrutura Geral da Qualidade de Serviço

[Lu, 96] propõe um modelo simplificado de operações de QoS que aborda os seguintes tópicos:

- especificação da qualidade de serviço;
- negociação e renegociação da qualidade de serviço;
- garantias da qualidade de serviço.

A aplicação especifica seus requisitos de QoS e os submete ao sistema. O sistema, a partir da especificação da QoS requerida, determina se existem recursos necessários para satisfazer os requisitos desejados. Em caso afirmativo, ele aceita a aplicação e reserva os recursos. Caso contrário, o sistema pode rejeitar a aplicação ou sugerir uma QoS menor, ou seja, com menos recursos solicitados.

3.2.1. Especificação da Qualidade de Serviço

Para fornecer especificações e garantias de QoS geralmente é utilizada uma sessão orientada à conexão. Antes do estabelecimento da conexão, os parâmetros de QoS devem ser especificados e negociados com todos os subsistemas interessados.

[Lu, 96] define um modelo de QoS considerando 3 camadas como ilustrado abaixo:

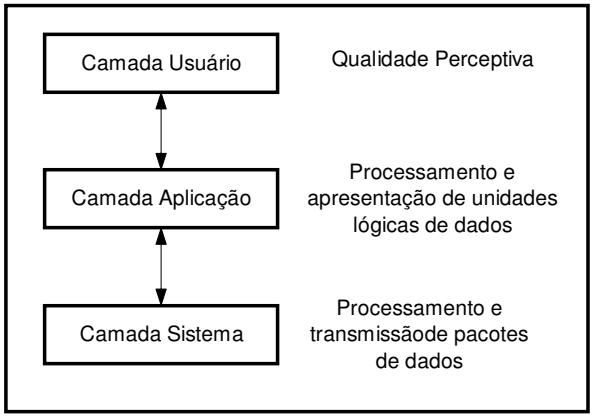

Figura 1 – Um modelo conceptual de QoS [Lu, 96]

3.2.1.1. Camada do Usuário

O resultado da QoS é a qualidade percebida pelo usuário final. Normalmente, o usuário final é aquele que dá início a QoS. Neste nível, a qualidade é normalmente medida qualitativamente, tal como excelente, bom, aceitável, não aceitável, ou muito pobre. A qualidade percebida é então algo subjetivo. A qualidade previamente escolhida pelo usuário implica diretamente na carga de serviço prestado; quanto maior a qualidade requerida, maior a carga. Este fato desencorajará os usuários a sempre escolherem a melhor qualidade.

Os usuários serão então o ponto de partida para uma consideração global de QoS. Assim, a fonte primária dos requisitos de QoS é o usuário e uma interface apropriada que deve ser fornecida para facilitar a escolha dos parâmetros. Neste nível, muitos parâmetros poderiam não ser entendidos pelo usuário e deveriam ser ocultos. Uma melhor abordagem é apresentar escolhas a partir de exemplos de diferentes qualidades, tal como vídeo de qualidade, TV normal ou HDTV, ou áudio de qualidade, telefone ou CD. A escolha do usuário é automaticamente mapeada em parâmetros da camada da aplicação.

3.2.1.2. Camada da Aplicação

As escolhas feitas pelo usuário são mapeadas em um conjunto de parâmetros que o nível de aplicação deve satisfazer para cumprir os requisitos do usuário. Os parâmetros neste nível são associados às unidades lógicas de dados, tal como quadro de vídeo e amostras de áudio. Para vídeo, alguns parâmetros típicos são: tamanho de imagem, bits por *pixel* e taxa de imagens. Para áudio, alguns parâmetros típicos são: taxa de amostragem e bits por amostra. Além disso, relacionamentos entre áudios, vídeos e outras mídias devem também ser especificados quando duas ou mais mídias relacionadas são usadas. Abaixo são mostradas algumas aplicações com a qualidade especificada pelo usuário e os parâmetros de QoS correspondentes no nível de aplicação.

Especificação do Usuário	**Parâmetro de Aplicação** **Taxa de amostragem**	**Parâmetro de Sistema** **Taxa de bits**
Qualidade de Voz Telefone	Amostragem = 8 kHz 8 bits por amostra	64 Kbits/s (s/ compactação) 16 Kbits/s (c/ compactação) Atraso fim-a-fim < 150ms Perdas de pacote < 1%

Áudio CD	Amostragem = 44.1 kHz 12 bits por amostra 2 canais	1.41 Mbits/s (s/ compactação) 128 Kbits/s (c/ compactação) Atraso fim-a-fim < 150ms Perdas de pacote < 0,01% Distorção entre 2 canais < 11 ms
Vídeo NTSC	30 fps resolução 720x480	200 Mbits/s (s/ compactação) 2 Mbits/s (c/ compactação)
HDTV	30 fps resolução 1440x1152	800 Mbits/s (s/ compactação) 10 Mbits/s (c/ compactação)
Sincronização labial	Distorção intermídia < 400 ms	Variação de Atraso Requisitos de Buffer

Tabela 3 – Aplicações e parâmetros de QoS

3.2.1.3. Camada do Sistema

Na camada de sistema, os parâmetros de QoS devem estar associados com as propriedades dos pacotes ou bits, tal como taxa de bits, taxa de pacotes e atraso de pacotes.

A coluna 3 da tabela acima mostra alguns exemplos de parâmetros a este nível. O sistema deve satisfazer estes parâmetros para cumprir os requisitos da aplicação. Esta é uma camada composta, que inclui o sistema operacional, protocolo de transporte, armazenamento secundário e rede básica. Portanto, os parâmetros podem ser adicionalmente decompostos. Parâmetros nestas sub-camadas podem ser diferentes e devem ser passados de uma camada para outra. Por exemplo, parâmetros na camada de transporte são associados com unidades de dado de protocolo de transporte. A camada de rede é interessada em pacotes de rede e o armazenamento em disco manipula dados em blocos.

Na camada de sistema são especificados os parâmetros fim-a-fim, tal como atraso e variação de atraso fim-a-fim. Durante a execução, estes parâmetros devem ser divididos em sub-requisitos que devem ser satisfeitos por subsistemas individuais. Por exemplo, se o atraso total fim-a-fim para uma aplicação é 100ms, então este atraso deve ser dividido em 30ms para um sistema-final, 40ms para a rede, e 30ms para o outro sistema final. Estes atrasos devem ser adicionalmente divididos tanto quanto necessário. Esta subdivisão dos parâmetros fim-a-fim é um problema complexo e deve ser feito durante o processo de negociação.

Geralmente os mapeamentos entre camadas arquiteturais não são um-a-um. Alguns parâmetros são mutuamente dependentes ou contraditórios. Por exemplo,

reduzindo a taxa de erros pela permissão de retransmissão, aumenta o atraso de trânsito médio. Além disso, na prática, os valores de QoS requeridos não correspondem a um ponto bem definido, mas a uma região no espaço do parâmetro. O ponto de trabalho instantâneo dentro desta região pode mudar no tempo.

3.2.2. Negociação e Renegociação de QoS

Durante o processo de negociação, os seguintes passos são realizados:

- os parâmetros de QoS são passados e negociados de uma camada ou um subsistema para outro;
- cada camada ou subsistema deve determinar se ele pode suportar o serviço requerido. Em caso afirmativo, certos recursos são reservados para a sessão. Quando todos os subsistemas aceitam os parâmetros de QoS a sessão é estabelecida. Senão a sessão é rejeitada. Neste último caso, sistemas sofisticados podem indicar ao usuário qual nível de QoS pode ser suportado. Se o usuário está contente com o nível de qualidade sugerida, a sessão é estabelecida.

As comunicações multimídia não são estáticas. Durante uma sessão ativa, ocorrem trocas na QoS por várias razões:

- usuário decide reduzir a qualidade de apresentação ou eliminar certos canais;
- usuário decide aumentar a qualidade de apresentação;
- necessidade de um canal extra para acessar informações multimídia adicionais.

Portanto, é necessário fornecer mecanismos de renegociação para satisfazer trocas de requisitos de comunicações multimídia. Algumas vezes não é possível satisfazer requisitos para aumentar a QoS porque os recursos requeridos podem não estar disponíveis.

3.2.3. Garantias de Qualidade de Serviço

Em geral, existem três níveis de garantia de qualidade de serviço:

- *Garantia determinista ou rígida:* é mais custosa em termos de recursos, pois os recursos são alocados a 100% e eles não podem ser usados por outras aplicações mesmo quando não estão sendo utilizados, o que resulta em um baixo uso dos recursos;

- *Garantia estatística ou soft:* é mais apropriada para mídias contínuas, pois não necessitam da utilização dos 100% dos recursos na apresentação. O uso destes recursos é mais eficiente, pois esta garantia é baseada em multiplexação estatística, onde os recursos não utilizados por uma aplicação podem ser usados por outras;

- *Melhor esforço:* neste caso, nenhuma garantia é fornecida e a aplicação é executada com os recursos disponíveis. Os sistemas computacionais tradicionais operam neste modo. A QoS pode ser garantida apenas quando recursos suficientes são disponíveis e o escalonamento de processos é apropriadamente implementado. A prevenção de sobrecargas requer controle de admissão e a prevenção de que aplicações não utilizem mais recursos do que aquele alocado requer mecanismos de policiamento.

Para fornecer garantias de QoS, técnicas de gestão de recursos devem ser usadas. Sistemas multimídia não podem fornecer QoS confiável aos usuários sem a gerência de recursos (ciclos de processamento de CPU, largura de banda da rede, espaço em buffer nos comutadores e receptores) nos sistemas finais, rede e comutadores. Sem a reserva de recursos, atrasos ou corte de pacotes devido a não disponibilidade de recursos necessários podem acontecer na transmissão de dados multimídia.

Para garantias de QoS fim-a-fim é necessário que, cada subsistema disponibilize funções de gerenciamento de QoS, incluindo cinco elementos: especificação de qualidade de serviço, controle de admissão, negociação e renegociação, alocação de recursos e policiamento de tráfego.

3.3. Protocolo de Reserva de Recurso (RSVP)

Para possibilitar o gerenciamento de QoS em nível de redes é necessária a existência de um protocolo de reserva de recurso. Este tipo de protocolo, na realidade, não executa a reserva do recurso em si, ele é apenas o veículo para transferir informações acerca dos requisitos de recursos e é usado para negociar os valores de QoS que o usuário deseja para suas aplicações. Para a reserva de recursos, os subsistemas devem prover funções de administração de recurso que forcem e escalonem acessos a recursos durante a fase de transmissão de dados.

O RSVP (*Resource Reservation Protocol*) [Zhang, 94] é um protocolo projetado para aumentar o suporte para aplicações tempo-real em redes IP. Ele permite a reserva de recursos em um caminho, mas a transmissão de dados é de responsabilidade do IP. Neste sentido, ele deve ser visto como um protocolo companheiro do IP.

No RSVP, a QoS não é especificada para a rede pelo emissor da informação, mas pelo receptor. A idéia é que o receptor está mais bem colocado que o emissor para saber que a QoS é necessária. Por exemplo, não há necessidade de que o emissor envie um fluxo de vídeo a 6 Mbps para o receptor se ele não tem poder de processamento para decodificar mais que 3 Mbps. A implicação desta diferença é que RSVP é mais eficiente no uso de recursos da rede, pois é reservado o estritamente utilizável, além de permitir requisitos de receptor heterogêneos.

Resumidamente, o mecanismo de reserva do RSVP trabalha da seguinte forma:

- A aplicação fonte envia regularmente mensagem especial chamada *path* para um endereço *multicast*. Esta mensagem contém a especificação do fluxo. Esta mensagem estabelece o estado *path* nos agentes RSVP intermediários que são usados na propagação dos pedidos de reserva (feita pelos destinatários) para uma fonte específica;

- Na recepção da mensagem *path*, cada receptor usa informações desta mensagem e informações locais (recursos computacionais, requisitos da aplicação, restrições de custo) para determinar a QoS. Em seguida ele

responde a mensagem *path* por uma mensagem *Reservation,* especificando a QoS requerida;

- A rede reserva recurso no caminho de retorno da mensagem *Reservation* para a aplicação fonte. Na passagem da mensagem *Reservation,* os agentes intermediários reservam recursos de rede ao longo do caminho e usam o estado *Path* estabelecido para propagar o pedido para o grupo emissor. A propagação da mensagem *Reservation* termina quando o caminho emenda em uma árvore de distribuição com recursos alocados suficientes para satisfazer os requisitos pedidos.

Quando a QoS exigida por um receptor difere do fluxo emitido pela fonte, filtros de tráfego são usados para reduzir os requisitos de QoS nos agentes RSVP apropriados. Por exemplo, se um receptor é apenas capaz de apresentar imagens preto&branco e a fonte libera dados de imagens coloridas, um filtro será usado para remover os componentes de cor. O filtro também pode combinar vários fluxos de dados em um, antes de enviar ao receptor. Portanto, o estilo de reserva iniciado pelo receptor acomoda requisitos heterogêneos dos receptores. Além de preservar a largura de banda da rede.

O protocolo RSVP utiliza outros protocolos para efetuar roteamento e transmissão. Seu objetivo único é a reserva, manutenção e liberação de recursos quando solicitado. Por isso, pode operar em *unicast*, *multicast*, Ipv4, Ipv6, e outros. A figura 7 mostra os módulos necessários em uma implementação RSVP, e em seguida tem um resumo deles:

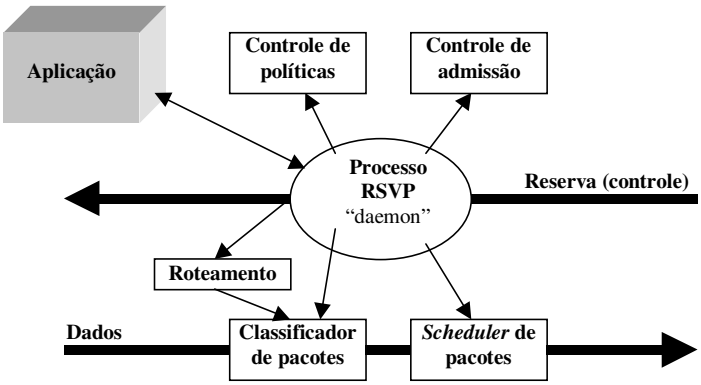

Figura 2 – Módulos necessários em uma implementação RSVP

- *Controle de admissão:* utilizado no início da chamada para verificar se o nó tem recursos suficientes para atender a qualidade de serviço solicitada. Para aceitar uma nova comunicação ou conexão, é necessário haver mecanismos que verifiquem se o sistema possui os recursos suficientes para atender aquele pedido. Isso deve ser realizado no estabelecimento de uma nova conexão ou na modificação de uma conexão já em operação, e toma como base os parâmetros definidos na especificação do fluxo de dados;

- *Controle de políticas:* determina se o usuário tem permissão administrativa para fazer a reserva;

- *Classificador e escalonador de pacotes:* o classificador de pacotes determina a classe de QoS. Quando a solicitação passa pelo controle de admissão e políticas, são configurados alguns parâmetros nesses módulos, a fim deles reconhecerem os pacotes para ordenar corretamente na saída, dando a necessária qualidade de serviço para cada fluxo;

- *Fluxo de dados:* RSVP é simplex, ou seja, faz reservas para fluxos unidirecionais. Para que o sistema seja capaz de reservar recursos, é necessário que haja uma especificação do comportamento do fluxo que será trafegado pela aplicação, assim como dos recursos necessários para este.

O processo do RSVP também se comunica com as rotinas de roteamento para determinar o caminho das solicitações de reserva. Isso causa um problema no caso de mudança de uma rota na tabela de roteamento (que é dinâmica), gerando uma necessidade de reserva (feita automaticamente) através do novo caminho. Para conseguir isso, o caminho estabelecido é do tipo "*soft state*", necessitando mensagens periódicas para se manter. Na ausência de tais mensagens (por uma mudança de rota ou saída de cliente), a interface dá *time-out* e a reserva é liberada.

3.3.1. A Operação do Protocolo

O fluxo de informação no RSVP pode ser subdividido em três categorias:

- Os dados RSVP gerados pela fonte de conteúdo (transmissor), especificam suas características de tráfego (*sender TSpec*) e os parâmetros de QoS associados (*sender RSpec*). Essa informação é transportada, sem qualquer alteração, pelos elementos de interconexão de redes num objeto RSVP *SENDER_TSPEC* até o(s) receptor(es). Um RSVP *ADSPEC*, também é gerado pela fonte de conteúdo e leva informações descrevendo as propriedades do caminho dos dados (*data path*), incluindo a disponibilidade de serviços específicos de QoS;

- Os dados RSVP gerados pelos elementos da rede de interconexão (normalmente, roteadores e comutadores ATM), são usados pelos receptores para determinar quais recursos estão disponíveis na rede. Os parâmetros de QoS que são reportados, podem ajudar os receptores a determinar a banda disponível, os valores de retardo nos links e os parâmetros de operação. Tal qual nas fontes de conteúdo, um objeto RSVP *ADSPEC* pode ser gerado pelos elementos de interconexão de rede, o qual transporta uma descrição dos serviços disponíveis de QoS. A existência de dois objetos, um *ADSPEC* e um *SENDER_TSPEC*, descrevendo os parâmetros de tráfego para *downstream* (em direção ao receptor) pode ser confuso. No entanto, existe uma distinção. O *SENDER_TSPEC* contem informações que não podem ser modificadas, enquanto o conteúdo *ADSPEC* pode ser atualizado na rede;

- Os dados RSVP gerados pelo receptor, especificam as características de tráfego da descrição dos pacotes (*receiver TSpec*) e uma perspectiva de recursos (*receiver RSpec*). Esta informação é colocada num RSVP *FLOWSPEC* e transportada num fluxo *upstream* através dos elementos de interconexão de rede, até a fonte de conteúdo. Ao longo do caminho, devido a concatenação das reservas, é possível modificar o *FLOWSPEC*.

4. Sistemas Operacionais

Um sistema operacional é um programa que atua como uma interface entre o *hardware* do computador e o usuário do sistema. Seu propósito é fornecer um ambiente no qual se possa executar programas. Suas metas são tornar o sistema do computador conveniente ao uso e que a utilização do *hardware* seja feita de modo eficiente.

4.1. A Função do Sistema Operacional

O SO é um programa que controla e coordena o uso do *hardware* do computador entre os vários programas de aplicação para os vários usuários [Silberchatz, 00]. Assim, podemos dizer que o SO é um conjunto de módulos de *software* que regem os recursos do sistema, resolvem seus conflitos, simplificam o uso da máquina e otimizam seu desempenho global.

Um SO deve ter o completo domínio sobre os recursos da máquina e exercer várias funções, dentre elas: escalonamento de recursos, controle de entrada e saída (E/S), gerência da memória, gerência do processador, segurança e escalonamento de processos.

No projeto de um SO deve-se ter em mente dois objetivos principais: apresentar ao usuário do computador uma forma amena de utilizar a máquina criando uma máquina virtual, de fácil compreensão para o usuário, com características diferentes da máquina física e realizar o melhor uso possível do *hardware* disponível, aumentando o desempenho do sistema e diminuindo o custo.

4.2. Área de Aplicação

É possível dividir as áreas de aplicação de um SO em sistemas de tempo real e sistemas de processamento de tarefas.

4.2.1. Sistemas de Tempo Real

Os sistemas de tempo real são aqueles que devem fornecer uma resposta a estímulos externos num período de tempo extremamente pequeno. Como exemplo, temos:

- *Controle de processos:* os computadores são utilizados para controlar processos industriais tais como o refino de petróleo bruto, controle de temperatura em altos fornos entre outros. Em comum, tais aplicações têm a necessidade de receber uma resposta rápida após a emissão de um sinal de controle;

- *Consulta a base de dados:* o computador é utilizado para obter informações armazenadas em grandes bancos de dados. Geralmente, o usuário deste tipo de aplicação desconhece como se processam as operações do sistema, e espera um tempo de resposta pequeno para obter suas informações. Um exemplo de tal aplicação seria uma consulta a informações sobre censo;

- *Processamento de transações:* neste caso, o computador é utilizado para realizar acessos a bancos de dados que estão freqüentemente sendo atualizados, talvez várias vezes em poucos segundos. As aplicações típicas são reservas de passagens e consultas bancárias.

4.2.2. Sistemas de Processamento de Tarefas

Os sistemas de processamento de tarefas são projetados para manipular um fluxo contínuo de programas que serão executados pelo computador. Tendo em vista a grande variedade de programas que podem ser processados, tal sistema precisa suportar um grande número de utilitários. Esses utilitários podem ser compiladores para diversas linguagens, montadores (*assemblers*), editores de texto entre outros. É necessário também, dar suporte a um sistema de arquivos para gerenciar o armazenamento de informações.

Podemos classificar os sistemas de processamento de tarefas em dois grupos:

- *Batch:* a principal característica desse grupo é o fato de que o usuário perde o controle do programa a partir do momento em que ele o submete ao sistema;

- *Interativo:* a característica marcante desse grupo é permitir a monitoração e o controle do programa, através de um terminal, enquanto durar o processamento.

A Internet fornece às aplicações multimídia um serviço único do tipo melhor esforço. Sendo assim, aplicações em tempo real e aplicações que não ocorrem em tempo real recebem o mesmo tratamento no nível do suporte de comunicação.

4.3. Serviços do Sistema Operacional

Uma visão bastante comum do SO é aquela que encara este *software* como uma extensão da máquina, fornecendo mais serviços para os aplicativos e outros programas básicos. Ele cuida de todos os recursos que estão disponíveis no computador, permitindo ao usuário utilizar a máquina (*hardware* + SO) de maneira amigável.

4.3.1. Tipos de Serviços

O SO fornece um ambiente para a execução de programas através de serviços dos próprios programas e para os usuários desses programas. Alguns serviços não têm como preocupação apenas tornar a máquina mais confortável para o usuário, mas também, para que o próprio sistema seja mais eficiente e seguro. Esse é o caso dos serviços oferecidos nos sistemas que permitem vários usuários compartilhando todos os recursos da máquina. Apesar da forma como esses serviços são oferecidos variar de sistema para sistema, existem algumas classes de serviços que são comuns a todos os sistemas operacionais. Estes compõem a sua própria definição. Como exemplo temos:

- *Execução de programas:* o SO é o responsável por carregar um programa na memória principal da máquina e executá-lo. O programa é o responsável pelo término da sua própria execução;

- *Operações de entrada/saída:* durante a sua execução, um programa pode ter necessidade de se comunicar com o meio externo à máquina. Esta operação recebe o nome de entrada/saída (E/S) e pode envolver qualquer dispositivo de E/S, como disco e impressora. Como um programa não

pode executar estas operações diretamente, o SO é o responsável por fornecer meios adequados para isso;

- *Manipulação de sistema de arquivos:* o SO é o responsável por gerenciar o sistema de arquivos da máquina. Este gerenciamento inclui a alocação de espaço no dispositivo de armazenamento secundário, a busca otimizada a um determinado arquivo e o armazenamento de todas as informações necessárias sobre cada arquivo;

- *Detecção de erros:* o SO é o responsável por detectar erros possíveis que podem comprometer a execução de qualquer programa e a segurança da máquina. Estes erros podem envolver o próprio processador, a memória principal (acesso a uma área proibida), os dispositivos de entrada/saída (falta de papel na impressora), ou até mesmo o programa do usuário (uma divisão por zero);

- *Alocação de recursos:* é de responsabilidade do SO a alocação dos diversos recursos em sistemas com um ou mais usuários. Estes recursos incluem a memória principal, a própria CPU, arquivos e os dispositivos de E/S. A alocação deve ser feita da forma mais eficiente possível para não prejudicar o desempenho do sistema;

- *Proteção:* o SO é o responsável pela proteção a todo o sistema computacional. Essa proteção se torna necessária tanto em sistemas monousuários quanto em sistemas multiusuários. A única diferença é a sua complexidade. Quando vários usuários estão usando o sistema a execução de um programa não pode interferir na execução de outro.

4.4. Sistema de Arquivos

Os dados e programas são armazenados em arquivos e esse armazenamento de informações possibilita a recuperação, reutilização e modificação nos dados e nos programas. Os computadores podem armazenar informações em vários dispositivos físicos diferentes, tais como fitas e discos magnéticos e óticos.

4.4.1. Conceito de Arquivos

Os arquivos podem ser definidos como uma unidade lógica de armazenamento de informação, destinada a abstrair as propriedades físicas dos meios de armazenamento. Ou ainda, uma seqüência de registros cujo significado é definido pelo seu criador.

Um arquivo é referenciado por seu nome e tem propriedades tais como tipo, tempo de criação, tamanho, nome do proprietário entre outras. Estas informações ficam armazenadas num diretório, que é uma tabela de símbolos que permite identificar tais informações. Ainda podem ser realizadas operações sobre os arquivos através de chamadas ao SO e são: criação, escrita, leitura, reposicionamento para o início e apagar.

4.4.2. Armazenamento

As formas físicas mais comuns que os sistemas de computadores utilizam para armazenar informações são fitas magnéticas e discos. Cada um destes dispositivos tem suas próprias características e organização física.

4.4.3. Gerência de Espaço em Disco

O sistema de arquivos deve ser capaz de controlar a área de espaço livre nos discos, utilizar métodos de acesso às informações armazenadas e métodos de alocação que sejam convenientes e eficientes.

As informações armazenadas em arquivos devem ser buscadas e colocadas dentro da memória do computador para serem utilizadas. Os métodos de acesso às informações de um arquivo podem ser feitos de modo seqüencial ou direto. Alguns sistemas fornecem somente um dos métodos, outros oferecem os dois.

4.5. Sistema de Diretório

No diretório são armazenados dois tipos de informação. A primeira está relacionada com o dispositivo físico (a localização do arquivo, seu tamanho e método de alocação). A segunda está relacionada à organização lógica dos arquivos (nome, tipo, proprietário, códigos de proteção).

O diretório é essencialmente uma tabela de símbolos. O SO utiliza o nome do arquivo simbólico para achar o arquivo. Quando considerarmos uma estrutura de diretório em particular, devemos ter em mente as operações que podem ser realizadas no diretório. Dentre estas operações podemos citar: busca, criação, apagar, listar seu conteúdo e fazer cópias para *backup***.**

Podem existir várias estruturas de diretório como:

- *Diretório em um único nível:* todos os arquivos estão contidos no mesmo diretório e seus arquivos devem ter nomes distintos;
- *Diretório em dois níveis:* cada usuário tem seu próprio diretório de arquivo de usuário;
- *Diretório estruturado em árvore:* nesta estrutura existe um diretório raiz da árvore onde os nós intermediários da árvore são os diretórios dos usuários, que podem ainda criar seus próprios subdiretórios e organizar seus arquivos;
- *Diretório em grafo acíclico:* permite que subdiretórios e arquivos sejam compartilhados, ao contrário da estrutura em árvore que não permite o compartilhamento explícito. Um grafo acíclico é uma generalização do esquema de diretório estruturado em árvore e não contém ciclos;
- *Diretório em grafo geral:* existem diversos caminhos que levam a um mesmo arquivo ou diretório e permite que subdiretórios formem ciclos.

4.6. Processos

Informalmente, um processo é um programa em execução. A execução de um processo se dá de maneira seqüencial, ou seja, em qualquer instante de tempo no máximo uma instrução está sendo executada.

Uma outra idéia associada à definição de processo, está na necessidade de descrever a existência de várias atividades que ocorrem em paralelo dentro do sistema computacional. Em um ambiente multiprogramado vários usuários podem estar executando

seus programas simultaneamente, dificultando a gerência de múltiplas atividades paralelas. Dessa forma, o modelo de processos é uma maneira de se decompor este problema em componentes mais simples. Todo o *software* no computador é organizado em processos seqüenciais ou apenas processos.

Com a multiprogramação, cada usuário tem a sensação de ter o processador só para si, mas na verdade, o processador central se reveza entre os vários usuários e uma das tarefas do sistema operacional é tornar isto o mais transparente possível.

Quando o processador muda de um processo para outro, é necessário que o sistema salve todas as informações necessárias para a retomada do processo interrompido. Dentre essas informações pode-se destacar: a identificação do processo, o seu estado, o valor do contador do programa, o valor dos registradores que estavam sendo usados (acumulador, de uso geral), as informações para gerência da memória (registradores de limite, endereço do início da tabela de páginas), as informações para escalonamento (prioridade, ponteiros para as filas de escalonamento) entre outras.

Assim, pode-se dizer que ao conceito de processo estão associadas informações que caracterizam o seu contexto de execução. Estas informações são armazenadas numa estrutura que recebe o nome de Bloco de Controle do Processo (*Process Control Block* - PCB) e que deve estar armazenada na área de memória destinada ao sistema operacional, para evitar que o usuário possa realizar o acesso a ele. Os PCB's representam os processos para o SO e a sua quantidade varia com o tempo à medida que processos são criados e terminam dinamicamente. Durante a execução de um programa o processo pode se encontrar em vários estados:

- *Executando:* se ele está usando o processador;
- *Pronto:* quando espera que a CPU seja liberada pelo processo que a está usando;
- *Bloqueado:* quando não pode ser executado porque espera que alguma condição externa aconteça;
- *Terminado:* quando a última instrução do programa foi executada.

4.7. Escalonamento da CPU

O escalonamento da CPU é o conceito mais relevante de sistemas operacionais multiprogramados. É através do chaveamento do processador entre os vários processos, que o SO pode tornar a máquina mais eficiente e produtiva.

A visão que o SO tem dos processos e seus respectivos estados pode ser representada por filas formadas pelos PCB' s. Assim, um processo que esteja no estado pronto é mantido numa fila de processos prontos. A forma como os processos são colocados e retirados dessa fila é discutida a seguir e é um dos pontos mais importantes na implementação do SO.

Quando um processo sai do estado executando e passa para o estado bloqueado é porque alguma interrupção ocorreu e alguma condição externa deve ser satisfeita. Ele é colocado na fila associada ao dispositivo que será usado durante a operação de E/S. Por exemplo, se o acesso for ao disco, o processo deverá ser colocado na fila deste dispositivo, caso ele esteja sendo usado por algum outro processo.

A escolha do processo que vai ser retirado de uma fila recebe o nome de escalonamento e é implementado por um componente do sistema operacional denominado escalonador.

Existem dois tipos de escalonadores principais: os de longo prazo e os de curto prazo. O primeiro é o responsável por escolher quais programas vão ser carregados na memória e passar para o estado pronto. Este escalonador é executado em intervalos de tempo maiores, já que um programa só libera a área de memória quando termina a sua execução. O segundo é o responsável por selecionar qual processo vai ser retirado da fila de prontos e ser executado na CPU. A grande importância dos escalonadores de longo prazo é que eles definem o nível de multiprogramação do sistema, controlando o número de programas carregados na memória ao mesmo tempo.

Um outro componente do sistema operacional envolvido no escalonamento da CPU recebe o nome de despachador. Ele é o responsável por passar o controle da CPU para o processo selecionado pelo escalonador de curto prazo. Esta operação envolve o carregamento dos registradores do processador com os dados do processo selecionado e o

desvio para o ponto do programa de onde a execução deve iniciar (nem sempre é do início, dado que durante a execução, um processo pode ser suspenso, pois pode ter que esperar por algum evento). Estas informações estão guardadas no PCB do processo selecionado. Além disso, o despachador deve ser o mais rápido possível já que influencia diretamente na utilização da CPU.

4.7.1. O Escalonamento de Processos

O escalonamento da CPU lida com o problema de decidir qual dos processos que se encontram na fila de prontos será o escolhido para executar. Existem várias maneiras de se implementar esta escolha, mas é importante saber que ela influencia na eficiência do SO. Como os algoritmos possuem propriedades diferentes, há necessidade de alguns critérios para se selecionar qual o melhor algoritmo a ser usado numa situação particular. Estes critérios são apresentados a seguir:

- *Utilização da CPU:* como a CPU é um recurso caro e influencia diretamente na eficiência da máquina, ela deve ser mantida ocupada o maior tempo possível;

- *Throughput (taxa de saída):* um modo de medir o trabalho realizado pelo sistema é contabilizar a quantidade de processos que terminam num determinado intervalo de tempo;

- *Turnaround time:* do ponto de vista dos processos, um critério importante é a quantidade de tempo que cada um dos processos passa dentro do sistema para executar. O *turnaround time* é contado a partir do momento em que o processo é submetido ao sistema até o instante em que a última instrução é executada. Inclui todos os intervalos gastos esperando na memória, esperando na fila de prontos, executando na CPU e realizando operações de E/S;

- *Tempo de espera:* o escalonamento da CPU não afeta a quantidade de tempo que o processo gasta executando no processador nem a quantidade de tempo gasto realizando E/S. Estes tempos dependem do próprio processador e dos dispositivos de E/S. Por outro lado, os algoritmos de

escalonamento afetam diretamente o tempo que os processos gastam esperando nas filas. Este tempo é medido através do tempo de espera;

- *Tempo de resposta:* em sistemas interativos (*on-line*) o *turnaround time* às vezes não é o melhor critério de medida. Para estes sistemas uma boa medida seria o intervalo de tempo entre a submissão de um pedido e a resposta obtida. Esta medida é o tempo de resposta.

Uma vez selecionado um algoritmo para implementar o escalonamento, é desejável que a taxa de utilização e a taxa de saída sejam maximizadas, e que o *turnaround time*, os tempos de espera e de resposta sejam minimizados. E ainda que todos os processos recebam o processador em algum momento, para que possam ser executados. Além disso, é muito importante que o SO atenda a estes objetivos mantendo total transparência para os usuários. A quantidade máxima de programas executando concorrentemente deve ser tal que não degrade a eficiência do sistema computacional.

Os algoritmos de escalonamento podem ser divididos em duas classes: preemptivos, onde o processo que está sendo executado pela CPU pode ser interrompido e ainda perder o controle do processador para outro processo mais prioritário e não-preemtivos, onde o processador não pode ser retirado do processo que está sendo executado, a não ser quando o seu ciclo de CPU terminar.

4.8. Gerência de Memória

É na memória principal que ficam armazenados os programas que serão executados e a maior parte dos dados que serão manipulados. A memória interage com a CPU e com o subsistema de entrada e saída.

O SO é responsável pelo controle das atividades da memória, tais como manter informações sobre a sua ocupação, armazenar e liberar espaços, decidir quais processos serão carregados quando houver espaços livres. Existem diferentes esquemas de gerenciamento de memória.

Num sistema de computador tanto a CPU quanto o subsistema de E/S interagem com a memória. Dado que cada conteúdo (palavra ou *byte*) armazenado na memória possui

seu próprio endereço, a interação é feita através de uma seqüência de leituras e escritas a endereços de memória específicos.

Em um sistema de computador existem tipos de memória com diferentes características, que formam uma hierarquia como a ilustrada na figura 4.

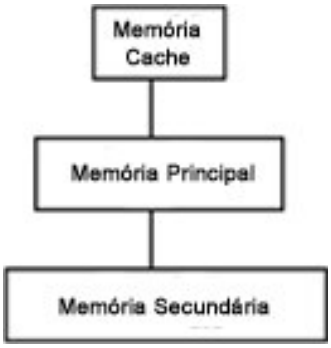

Figura 3 – Hierarquia de memória

A hierarquia de memória pode ser analisada segundo suas capacidades de armazenamento, custo por *bit* e tempo de acesso. Partindo do nível mais inferior da Figura 4, as memórias secundárias são capazes de armazenar uma grande quantidade de informação, seu custo por *bit* é menor e o seu tempo de acesso é relativamente maior do que as memórias dos níveis superiores.

No segundo nível, está a memória principal. Ela é capaz de armazenar uma quantidade menor de informação, seu custo por *bit* é maior e seu tempo de acesso é menor do que as memórias secundárias.

No nível mais superior estão as memórias *cache*. Estas memórias são as mais rápidas e com o maior custo por *bit*. Por serem muito caras as memórias *cache* são pequenas, isto é, são as que têm menor capacidade de armazenamento em relação as demais.

O gerenciamento de memória usa regras, ou políticas de gerenciamento para:

- *Busca:* determina quando um bloco de informação deve ser transferido da memória secundária para a memória principal;

- *Armazenamento:* determina onde o bloco de informação deve ser colocado na memória principal;

- *Substituição:* determina qual bloco de informação deve ser substituído por um novo bloco.

Em um sistema monoprogramado a gerência da memória principal é menos complexa que nos sistemas multiprogramados. Nos sistemas monoprogramados, uma parte do SO permanece sempre residente na memória, e a cada instante somente um único programa do usuário está carregado na memória principal. O compartilhamento da memória entre o SO e o programa do usuário, necessita que o sistema possua mecanismos de proteção. Estes mecanismos permitem verificar a validade dos acessos à memória gerados pelo programa do usuário para evitar danos ao SO.

4.8.1. Organização de Memória

É necessário que todo o espaço de endereço lógico de um processo esteja na memória física antes que o processo possa ser executado. Dessa forma, o tamanho de um programa fica restrito ao tamanho da memória física do sistema. Estas organizações são ditas organizações de memória real.

Pode-se organizar a memória em duas partições:

- *Partições Fixas:* nesta organização a memória é dividida em número fixo de partições ou regiões. A cada partição pode ser atribuído um processo para ser executado. Quando existe uma partição livre, um processo é selecionado de uma fila e carregado naquela partição. Quando ele termina sua execução, a partição torna-se livre para um outro processo;

- *Partições Variáveis:* o problema principal com a organização da memória em partições fixas é a determinação do tamanho das partições de modo que a fragmentação interna e externa seja mínima. A organização da memória com partições variáveis visa solucionar este problema permitindo que os tamanhos das partições variem dinamicamente.

Para a implementação da organização com partições variáveis, o SO mantém uma tabela indicando quais partes da memória estão disponíveis e quais estão ocupadas. A princípio, toda a memória está disponível e é considerada um grande bloco de memória. Quando um processo entra no sistema e necessita de memória, é realizada uma busca por um espaço que seja grande o suficiente para armazená-lo. Se existe tal espaço, é atribuído ao processo somente a quantidade de memória necessária. O restante do espaço, que pode haver, é deixado disponível para futuras requisições. Sempre que um processo termina sua execução ele libera seu espaço de memória. Este espaço liberado é colocado de volta junto com os espaços de memória disponíveis. Neste ponto, procura-se verificar se há áreas adjacentes que possam ser recombinadas de modo a formar espaços de tamanhos maiores.

As estratégias mais comuns para selecionar um espaço de memória são:

- *First-fit:* aloca o primeiro espaço grande o suficiente. Não necessita de uma busca por toda a lista. É a estratégia mais rápida;

- *Best-fit:* aloca o menor espaço que seja grande o suficiente. É necessário que se percorra toda a lista, a menos que esta esteja ordenada. Esta estratégia é a que provoca menor fragmentação da memória;

- *Worst-fit:* aloca o maior bloco. Também é necessário uma busca pela lista inteira, a menos que ela esteja ordenada por tamanho. Esta estratégia visa deixar livres espaços de memória maiores.

4.8.2. Memória Virtual

A técnica de memória virtual foi criada para permitir a execução de vários processos que não necessariamente estejam armazenados por inteiro na memória principal. Uma das vantagens mais importantes é que os programas dos usuários podem ser maiores do que a memória física. Além disso, como cada programa de usuário não necessita estar completamente armazenado na memória, podem ser executados um maior número de programas simultaneamente. Isto leva a uma maior utilização da CPU e um aumento no *throughput*.

A memória virtual é a separação da memória lógica do usuário da memória física. Isto torna a tarefa de programação mais fácil na medida em que o programador não precisa se preocupar com a quantidade de memória física disponível.

A implementação da memória virtual é comumente realizada com paginação sob demanda ou com segmentação sob demanda.

4.8.2.1. Swapping

A técnica de *swapping* requer que o sistema possua um *backing store*. O *backing store* é uma memória secundária, geralmente um disco rápido. Ele deve ser grande o suficiente para armazenar cópias dos programas de usuários e fornecer acesso direto a esses programas.

A fila de prontos consiste de todos os processos que estão no *backing store* e que estão prontos para serem executados. Um conjunto de variáveis do sistema indica quais processos estão correntemente na memória.

Sempre que o escalonador da CPU decide executar um processo, o despachador verifica se o processo escolhido está presente na memória. Caso não esteja, ele troca um dos processos correntes na memória pelo processo desejado. O despachador, então, é responsável pela troca de contexto dos processos. Isto é, salva o conteúdo dos registradores do processo que está sendo retirado, armazena nos registradores os valores correspondentes ao novo processo e transfere o controle para o processo.

O processo escolhido para ser retirado da memória e levado ao *backing store* deve estar completamente ocioso. Nenhum processo com entrada e saída pendente deve ser trocado.

4.8.2.1. Paginação

A paginação é uma organização de memória que permite que a memória física seja vista como se estivesse dividida em blocos de tamanho fixo, chamados *frames*. A memória lógica é também dividida em blocos do mesmo tamanho, denominados páginas. Quando um programa vai ser executado, suas páginas são trazidas do *backing store* e

carregadas nos *frames* disponíveis. O disco é dividido em blocos de tamanho fixo, cujo tamanho é o mesmo dos *frames* da memória.

O tamanho da página (e do *frame*) é definido pelo *hardware*. Tipicamente varia entre 512 e 2048 *bytes*, dependendo da arquitetura do computador.

4.8.2.2. Segmentação

A segmentação é um esquema de gerenciamento de memória que pode dar suporte à visão que o usuário possui sobre a memória. O usuário pensa na memória como um conjunto de sub-rotinas, procedimentos, tabelas, variáveis e assim por diante. Cada um destes segmentos possui um nome, é de tamanho variável e não possui uma ordenação específica. Os elementos dentro de cada segmento são identificados por sua posição dentro do segmento, como por exemplo, a primeira entrada numa tabela de símbolos. Dessa forma, um espaço de endereço lógico é uma coleção de segmentos. Os endereços especificam tanto o nome do segmento quanto a posição dentro do segmento. O usuário especifica então cada endereço por estas duas quantidades: um nome de segmento e uma posição.

Por simplicidade de implementação, os segmentos são numerados e referenciados por seu número. Normalmente, o programa do usuário é compilado e o compilador automaticamente constrói segmentos refletindo o programa de entrada.

Embora o usuário possa referenciar objetos dentro do programa por endereços bidimensionais, a memória física é uma seqüência unidimensional de palavras. Assim, é necessário definir um mapeamento entre estas duas visões. Este mapeamento é efetuado por uma tabela de segmentos.

O endereço lógico consiste de duas partes: um número de segmento *s* e um *offset* d, dentro do segmento. O número do segmento é usado como um índice para a tabela de segmentos. Cada entrada na tabela possui uma base e um limite do segmento. O *offset* d do endereço lógico deve estar entre 0 e este limite. Se for um acesso válido, o *offset* é adicionado à base do segmento para produzir um endereço físico. A tabela de segmentos então, é um *array* de pares de registradores base/limite.

4.8.2.3. Paginação Sob Demanda

A paginação sob demanda é similar a um sistema paginado com *swapping*. Os programas residem na memória secundária. Quando se inicia a execução, os programas são trazidos para a memória principal. Porém, nunca uma página é trazida para a memória se ela não for necessária. Com isso, diminuímos o tempo de troca e a quantidade de memória física necessária.

Para controlar o armazenamento das páginas trazidas para a memória, a tabela de páginas possui um *bit* de válido/inválido. Este *bit* é ativado quando a página está presente na memória. Se o programa tenta acessar uma página que ainda não foi trazida para a memória, então é gerada uma interrupção por falha de página. Neste caso, é utilizado o seguinte procedimento para carregar uma página busca-se um *frame* na lista de *frames* livres, escalona-se uma operação de disco para ler a página desejada para o *frame* alocado, quando a leitura do disco se completar, a tabela de páginas é modificada para indicar que a página agora está presente na memória e reinicia-se a instrução que foi interrompida pelo acesso ilegal.

4.8.2.4. Estratégias de Substituição

As estratégias de substituição de páginas visam escolher uma página para ser trocada por outra, se não existirem *frames* livres. Um *frame* é liberado escrevendo seu conteúdo num disco e trocando a tabela de páginas para indicar que a página não está mais residente na memória. O *frame* liberado pode ser então utilizado por outra página.

Existem diversos algoritmos de substituição diferentes. A escolha de um, em particular, leva em consideração a taxa de falha de páginas. Devemos escolher aquele que apresentar a menor taxa. Uma quantidade de trocas de páginas elevada degrada o desempenho do sistema.

A seguir, alguns algoritmos de substituição de páginas:

- *First-In-First-Out (FIFO):* é o mais simples. Associa a cada página, o tempo em que ela foi trazida para a memória. Quando uma página tiver que ser substituída, a página mais antiga na memória é escolhida;

- *Optimal replacement (OPT):* é o que apresenta a menor taxa de falhas de página. Neste algoritmo, é substituída a página que não será utilizada pelo maior período de tempo. Infelizmente, o algoritmo de substituição ótimo é difícil de implementar, uma vez que ele requer um conhecimento futuro das referências à memória. Por este motivo, o algoritmo ótimo é utilizado principalmente em estudos comparativos;

- *Least Recently Used (LRU):* é uma tentativa de aproximação ao algoritmo ótimo. O algoritmo ótimo utiliza na sua concepção o conhecimento futuro das referências à memória. O algoritmo LRU utiliza o conhecimento da história do passado recente das referências à memória, como uma aproximação do futuro. Este algoritmo associa a cada página seu último tempo de uso. Quando houver a necessidade de substituir uma página, é escolhida aquela que não foi utilizada pelo maior período de tempo. Esta estratégia é conveniente ao princípio da localidade. Por este princípio, quando uma página é referenciada, existe uma grande chance de que ela seja novamente referenciada em breve.

5. Estudo do Gerenciamento de QoS ao Nível de Sistema Operacional para Aplicações Multimídia Distribuídas

Aplicações multimídia, além de manipularem mídias discretas como texto, gráfico e imagem, também tratam de fluxos de mídia contínua, como áudio, vídeo e animação que, devido às suas características temporais, requerem gerenciamento de recursos e mecanismos de escalonamento diferentes daqueles utilizados pelos sistemas operacionais tradicionais. A correta execução de uma aplicação multimídia exige que os dados de áudio e vídeo, por exemplo, sejam percebidos pelos usuários de forma natural e com o mínimo de distúrbios, o que não é simples de ser garantido.

Encontrar garantias de QoS em sistema multimídia distribuídos é fundamentalmente uma questão de aplicação-para-aplicação. Isto normalmente requer, reserva de recursos e teste de admissão fim-a-fim em uma primeira instância, seguido por atividades do sistema operacional, como coordenação dos discos e controle de *threads*.

A observação principal é que para aplicações confiarem na transferência da multimídia, e em particular no fluxo contínuo da mídia, é essencial que a qualidade de serviço seja configurável, previsível e manutenível em todo sistema, incluindo dispositivos do sistema-fim, subsistema de comunicação e redes. Entretanto, é também importante que todos os elementos fim-a-fim da arquitetura dos sistemas distribuídos trabalhem em harmonia para alcançar o comportamento esperado da aplicação [Campbell 97].

Os sistemas operacionais com suporte a multimídia devem considerar as dependências lógicas e temporais existentes entre diferentes tarefas processadas no mesmo intervalo de tempo. Isto se refere, principalmente, ao processamento sincronizado dos fluxos de áudio e vídeo, no qual a relação temporal deve ser considerada. É exatamente esta dependência temporal existente entre várias mídias que sugere que o suporte oferecido pelo sistema operacional deva ser funcionalmente equivalente ao proporcionado pelos sistemas operacionais de tempo-real. No entanto, embora existam muitas similaridades entre as técnicas usadas em sistemas operacionais com suporte a multimídia e sistemas operacionais de tempo real, esta afirmação não é totalmente verdadeira.

5.1. SO de Tempo-Real X SO Multimídia

As tarefas de tempo-real são caracterizadas por um limite de tempo para o término da sua execução chamado de *deadline*. Para escalonar os processos de forma que suas *deadlines* sejam obedecidas, os sistemas de tempo-real requerem que a carga de processamento de cada tarefa seja previsível e, de certa forma, limitada. No entanto, tais condições dificilmente são estabelecidas para as aplicações multimídia, pois os processos de mídia contínua são executados no espaço do usuário, tornando complexa a definição exata da quantidade de processamento e de tempo de execução. Não é possível definir, por exemplo, o tempo gasto na descompressão de um quadro, embora possa ser estabelecido um limite.

Se um sistema operacional de tempo real convencional for usado para escalonar objetos multimídia, este pode funcionar bem desde que as cargas e tempos de processamento estejam dentro dos limites definidos. Mas se estes forem excedidos, o comportamento do sistema poderá se tornar imprevisível devido às perdas sucessivas de *deadlines*, principalmente porque os sistemas operacionais de tempo real não especificam o que deve ser feito em uma situação de sobrecarga de processamento, pois é assumido que tais situações não devem ocorrer.

Um sistema operacional com suporte a multimídia, por sua vez, não precisa se preocupar tanto em evitar a ocorrência de perdas. Na verdade, ele deve assumir que as perdas irão ocorrer, principalmente em situações de sobrecarga de processamento, e que nessas situações sua função é gerenciá-las de forma a minimizar o prejuízo, mantendo uma qualidade aceitável.

5.2. Gerenciamento de recursos e QoS

Um sistema computacional tem muitos recursos que podem ser requeridos para resolver um problema: CPU, memória de diferentes níveis, dispositivos de entrada e saída, disco, etc...

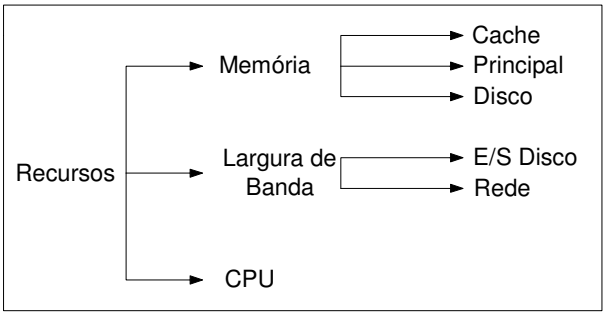

Figura 4 – Recursos do sistema operacional

Uma das principais funções de um SO é multiplicar esses recursos entre os usuários do sistema. No advento dos conflitos de recursos requisitados, o SO tradicional deve decidir para que requisitos serão alocados recursos para operar o sistema de computador de modo claro e eficiente [Peterson, 85]. Clareza e eficiência são ainda o mais importante objetivo do gerenciamento de recursos hoje em dia em sistemas operacionais. Entretanto, com respeito às aplicações multimídia, outros objetivos que são próximos se tornam de grande importância. Por exemplo, interação de usuários e sincronização requerem um curto tempo de resposta com limite superior e cadeias multimídia requerem um *throughput* mínimo por um certo período de tempo. Estes requisitos de aplicação são especificados como requisitos de QoS.

Tipicamente as especificações de QoS incluem tipos de parâmetros como taxa de quadro, resolução, atraso, atraso fim-a-fim e sincronização. Estes parâmetros de alto nível têm de ser mapeados em parâmetros e recursos de baixo nível que são necessários para dar suporte aos requisitos de QoS, como tempo de CPU por período, quantidade de memória e média e pico de largura de banda da rede.

Para encontrar os principais requisitos de QoS para aplicações multimídia é necessário gerenciar os recursos do sistema de tal maneira que os mesmos sejam disponíveis no tempo correto para executar a tarefa com a qualidade de serviço requerida.

Gerenciamento de recursos ao nível de sistemas operacionais, para englobar QoS, deve ter as seguintes especificações:

- *Especificação e alocação:* para recursos que são requeridos para executar a tarefa com a QoS solicitada;

- *Controle de admissão*: inclui um teste se os recursos disponíveis são suficientes para satisfazer a requisição sem interferir previamente com a condição solicitada. A maneira de este teste ser executado depende da especificação de requisitos e mecanismos de alocação usados para esse recurso;

- *Alocação e mecanismos de escalonamento*: asseguram que um eficiente compartilhamento dos recursos seja disponível no tempo correto. O tipo de mecanismo depende do tipo de recurso. Recursos que podem exclusivamente ser usados por um simples processo em um tempo tem de ser multiplicados no domínio temporal. Em outras palavras, recursos exclusivos, como CPU ou E/S de disco, tem de ser escalonáveis. Basicamente pode-se diferenciar entre escalonamento moderado, escalonamento de tempo-real e trabalhar ou não trabalhar conservando mecanismos de escalonamento;

- *Adaptação*: pode ser iniciada pelo usuário/aplicação ou pelo sistema que pode querer diminuir a QoS conforme os recursos solicitados, ou atualizá-los.

Especificação, controle de admissão, alocação e escalonamento, dependem do tipo de recurso, enquanto que adaptação e quantidade de recursos representam mais tipos de recursos de princípios independentes.

5.3. Gerenciamento de Memória

Os seguintes tópicos são as exigências em relação ao gerenciamento de memória para suportar garantias de QoS na execução de aplicações multimídia distribuídas ao nível de sistemas operacionais:

- *Critérios de Admissão:* as aplicações novas podem ser admitidas somente quando suas exigências de buffer de memória, junto com as alocações atuais do buffer de outros processos, não excedem um ponto inicial do total da memória disponível;

- *Garantias de Tempo-Real*: alguns mecanismos para assegurar garantias de tempo-real incluem:

    - *Latência de acesso limitada*: as aplicações críticas dos meios do tempo necessitam que o tempo de acesso à memória seja mínimo. Com memória virtual, é importante ter os mecanismos de paginação que têm um limite superior aceitável na latência do acesso;

    - *Semântica de cópia mínima*: os mecanismos como memória compartilhada podem ser usados para minimizar a sobrecarga entre *threads* co-relacionadas;

- *Clareza dos critérios* : Isto assegura a disponibilidade mínima de buffers de memória para todas as classes de aplicações multimídia. Isto poderia ser assegurado possivelmente calculando as dimensões dos buffers (durante a execução do sistema) para todas as classes de aplicações;

- *Crirérios de Manutenção e políticas*: Os critérios de manutenção requerem o ajuste de renegociação ou deixar sair uma solicitação (controle da admissão) no caso de sobrecarga de buffers. Policiar os critérios pode requerer a ofensiva da aplicação, que pode pedir mais do que sua requisição aos buffers compartilhados, seja notificando para uma renegociação mais adicional, ou em caso extremo, termina a sessão (após ter liberado todos os seus buffers).

5.4. Gerenciamento da CPU

Um bom sistema de reserva deve aceitar vários níveis de políticas de gerenciamento de recurso. Por exemplo, um gerenciador de QoS pode ser utilizado para alocação dos recursos do sistema, como um mecanismo para controlar os recursos alocados para as várias aplicações. O gerenciador de QoS transmite os parâmetros de QoS para as aplicações com suas requisições de recurso do sistema, incluindo as requisições de processador, possibilitando uma cooperação da aplicação em si [Yau, 96]. O gerenciador então é capaz de efetuar a reserva para cada processo e utilizar o desempenho analisado e,

através da interação com o usuário, mudar as alocações das aplicações sob o seu controle. Esta estrutura baseada nos requisitos de QoS divide o problema de escalonamento em duas partes: uma política para alocação dos recursos baseada nos requisitos da própria aplicação e uma abstração e mecanismo para escalonamento e controle destes recursos.

A estratégia da alocação precisa de mecanismos para possibilitar este tipo de gerenciamento de recursos. Os requisitos são:

- Prover algum meio para as aplicações especificarem suas alocações de recurso;
- Avaliar as requisições do novo processo para decidir se admite ou não;
- O escalonamento deve ser consistente com as políticas de controle de admissão;
- Medir exatamente o tempo de CPU consumido por processo, para assegurar que não ultrapasse suas alocações.

O primeiro requisito depende de um modelo de escalonamento consistente que possa acomodar diferentes espécies de alocações de processos. Por exemplo, uma aplicação de áudio precisa ser escalonada a cada 50ms para montar o buffer de áudio. Muitos processos não possuem requisitos de tempo e podem executar seguidamente tanto quanto possível para fazer sua execução.

A fração do processador permite uma medida para descrever a reserva do processador para as duas espécies de processos. Esta fração é um tempo do processador requerido pelo processo durante um intervalo de tempo dividido pelo tempo real deste intervalo e a fração do processador consumida pelo processo todo o tempo de sua taxa de progresso.

Os processos periódicos, ou seja, que executam repetidamente a um intervalo fixo, têm uma taxa natural descrita pelo seu período e o tempo de CPU requerido durante cada período. Nesse caso deve-se assumir o tempo de CPU constante. Caso esse tempo não seja constante, uma reserva do processador é estimada para o seu pior caso. O tempo ocioso é disponibilizado para o processamento *background*. Os processos aperiódicos não

têm a taxa natural de processamento mas devem receber uma. Essa taxa determina a duração de tempo até o processo terminar. A taxa precisa ser reservada baseada nos atrasos.

Quanto ao segundo requisito onde deve analisar as requisições do novo processo para decidir se admite ou não, o escalonador pode avaliar as restrições de tempo do novo processo diante da capacidade disponível. É transmitida à alocação requerida e especificada por um processo individualmente em suas medidas de utilização, que é utilizada pela política de controle de admissão. Observa-se que simplesmente somando-se a utilização individual de todos os processos ativos sob uma disciplina de escalonamento por prioridade dinâmica, à avaliação seria válida, mas há desvantagens e, em contrapartida, assumindo a utilização individual de todos os processos ativos sob uma disciplina de escalonamento por prioridade fixa, esse método não possibilita a reserva de 100% do processador [Mercer, 94].

O terceiro requisito, que a política de escalonamento escalone processos de forma consistente com a política de controle de admissão, remete em que a proposta precisa ser consistente com todas as políticas de gerenciamento de todos os recursos no sistema. Se a política de escalonamento não garante as premissas da política do controle de admissão sobre como os processos são ordenados para execução, a operação de alocação do sistema falha.

O quarto requisito demanda um desempenho rígido de monitoramento de software, o qual, um sistema operacional típico não suporta. Os sistemas operacionais usualmente acumulam estatísticas para cada processo, simplesmente durante uma interrupção regular do *clock*, mas essa informação é imprecisa sobre intervalos pequenos. Entretanto, o comportamento de execução do programa monitorado precisa ser independente de um simples período. Um mecanismo mais apurado se faz necessário para que se possa medir as durações entre troca de contexto e tempo de interrupções na sobrecarga do sistema.

Um algoritmo baseado na taxa de progresso do processo provê um ambiente efetivo para a implementação da alocação do processador. Pode-se associar taxas com processos periódicos e com não periódicos. A taxa de periódicos pode ser determinada por um período que o usuário tem em mente e o tempo de CPU durante este período. A taxa

para os processos não periódicos originam dos requisitos de atraso. Em qualquer caso, só a taxa não é suficiente para cumprir os requisitos de tempo de um processo. O tempo de CPU e o período são essenciais. Têm-se três valores que descrevem os requisitos do processador para um processo e, dois deles são requeridos para especificar a percentagem do processador.

O tempo de CPU de uma atividade periódica é difícil para o usuário medir exatamente. O ideal seria ter uma estimativa dada pelo usuário e então, dependendo do tempo medido pelo sistema em relação a aquele, pudesse ser ajustado se necessário.

O atraso para programas não periódicos que executam a uma dada taxa, pode ser calculado da taxa de execução e do tempo total de execução. Um processo que executa a uma dada taxa e com um total de tempo de CPU teria tempo suficiente para completar sua execução.

Assumindo que se pode determinar estes três parâmetros, as políticas de escalonamento tradicionais por prioridade fixa e dinâmica podem ser utilizadas pois permitem as alocações e o controle de admissão [Mercer, 94].

5.5. Adaptação

Existem duas motivações para adaptação em sistemas multimídia: requerimento de recursos é difícil de predizer (vídeo e interatividade); e a disponibilidade de recursos não pode ser garantida se o sistema incluir subsistemas de melhor-esforço (internet atual e sistemas móveis). Em sistemas de melhor-esforço, é somente possível adaptar a aplicação, relativamente à quantia de aplicação de dados que o sistema tem para tratar. Em sistemas operacionais, ambas as situações podem ocorrer e é possível adaptar ambas as aplicações e alocar recursos.

Em [Gecsei, 97] duas formas de adaptação são distinguidos, adaptação com *feedback* e adaptação sem *feedback*. A adaptação sem feedback significa que as aplicações mudam somente a funcionalidade da interface do usuário e não mudam quaisquer parâmetros dos recursos. Entretanto, será considerado neste trabalho somente adaptação com *feedback*. O modelo abaixo mostra uma simplificada visão da colaboração entre o consumidor de recursos e o fornecedor de recursos (escalonador da CPU) [Lakshman 97].

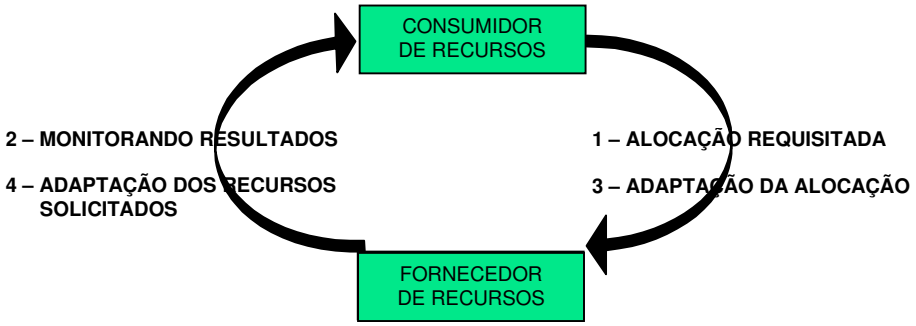

Figura 5 – Consumidor de recursos e fornecedor de recursos

1 – O consumidor de recursos, ou o gerente da entidade, estima sua solicitação de recursos e requisita ao fornecedor para alocar os recursos de acordo com sua especificação;

2 – Depois que o controle de admissão passar com sucesso, a utilização dos recursos é monitorada. O monitoramento dos resultados pode refletir a geral utilização dos recursos;

3 – O fornecedor solicita ao consumidor que ajuste seus requisitos de recursos, por exemplo, reduzindo a taxa de quadro de um vídeo;

4 – O consumidor solicita ao fornecedor que ajuste os parâmetros de alocação.

5.6. Escalonamento

Um escalonador pode ser definido como um processo utilizado para determinar a ordem de execução das tarefas, de modo que os requisitos do sistema em questão sejam obedecidos.

Para suportar os requisitos temporais, principalmente de mídias contínuas, o sistema operacional deve utilizar técnicas de escalonamento de tempo-real. Os mecanismos tradicionais de tempo-real usados para o controle de sistemas na área de automação de fábricas ou pilotagem de aviões, por exemplo, impõem restrições rígidas de segurança e tolerância à falhas. Estas exigências, no entanto, são bem mais flexíveis quando se trata de

aplicações multimídia. Na verdade, tais aplicações não se enquadram nestes cenários tradicionais de tempo-real, apresentando requisitos diferentes e mais favoráveis:

- Os requisitos de tolerância à falhas de aplicações multimídia são, usualmente, menos rigorosos que aqueles de sistemas de tempo-real que possuem impacto físico direto. O atraso na apresentação de quadros de vídeo, por exemplo, geralmente não causa a destruição de um equipamento ou a ameaça a alguma vida humana (exceto em algumas situações específicas, como aplicações de suporte a uma cirurgia remota);

- Para muitas aplicações multimídia a perda de uma *deadline* não representa uma falha muito severa, embora deva ser evitada. Um quadro de vídeo, por exemplo, não disponível no momento da apresentação, pode simplesmente ser omitido. Requisitos de áudio, entretanto, são mais rigorosos, já que falhas sonoras são mais perceptíveis pelo ser humano do que falhas visuais;

- Uma seqüência de dados digitais contínuos é, na verdade, o resultado de amostragens periódicas de mídia contínua. Esta periodicidade repete-se no processamento dos dados, o que facilita o escalonamento, quando comparado com a situação de tarefas esporádicas;

- A necessidade da largura de banda de mídia contínua não é sempre severa. Alguns algoritmos de compressão podem empregar diferentes taxas de compressão para qualidades diferentes e a largura de banda requerida pode ser negociada. Se a largura de banda que está disponível não é suficiente para a qualidade total, a aplicação pode aceitar uma redução dessa qualidade. A qualidade pode também ser ajustada dinamicamente para a largura de banda disponível, mudando parâmetros de codificação, por exemplo. Isto é conhecido como vídeo escalável.

Além disso, o escalonamento de objetos multimídia deve considerar os requisitos de QoS especificados para cada dado de mídia, incluindo as situações de insuficiência de recursos. Diante disso, diferentes medidas podem ser tomadas:

- *Mudar a funcionalidade das aplicações:* a mudança do som de estéreo (2 canais) para mono (1 canal) ou então, mudar a cor do vídeo de colorido para preto-e-branco;

- *Mudanças em relação à transmissão dos dados pela rede:* reduzir o tamanho do vídeo ou o tamanho da amostra de áudio através do uso de técnicas de compressão;

- *Mudar a freqüência na qual as aplicações são operadas:* diminuir a taxa de apresentação dos quadros do vídeo ou reduzir a taxa de amostragem do áudio.

As aplicações multimídia podem possuir requisitos temporais mais rígidos ou mais flexíveis, dependendo do seu tipo. Aquelas com características fortes de interação, como uma aplicação de videoconferência, por exemplo, exigem tempo de atraso da ordem de milisegundos, para que a aplicação não seja prejudicada. No entanto, na reprodução de um arquivo de vídeo os limites de atraso são mais flexíveis, embora as restrições de variância continuem as mesmas. O atraso na transferência de um único quadro do disco para o monitor não é tão significante. O usuário poderá perceber alguma diferença, mas entenderá como sendo apenas um atraso de resposta ao seu comando de iniciar a apresentação do vídeo. Em ambos os casos, um algoritmo de escalonamento viável para aplicações multimídia deve possuir as seguintes características:

- Conseguir um escalonamento viável, baseado nos requisitos de QoS da aplicação, mesmo em situações de sobrecarga de processamento;

- Utilizar um tempo mínimo de processador para as decisões de escalonamento, minimizando seu *overhead*;

- Assegurar, sempre que possível, que todas as tarefas que manipulam mídias contínuas sejam atendidas prioritariamente e terminem sua execução dentro de suas respectivas *deadlines*, mas sem deixar de atender as tarefas que manipulam mídias discretas.

Em máquinas que possuem apenas um processador apenas um processo pode ser executado em cada instante. Caso haja alguma requisição de dados não disponíveis na

memória e sim provenientes de entrada e/ou saída, esse processo deverá esperar o recebimento dos dados externos. A fim de evitar tal espera inútil (*busy waiting* ou espera ocupada), são utilizadas políticas de escalonamento. Assim, sempre que um processo tiver que aguardar eventos ou ocorrer uma interrupção, um outro processo se utilizará da CPU.

6. Análise de Sistemas Existentes

Esse capítulo mostra o estudo de alguns sistemas operacionais existentes onde foram analisados alguns recursos disponíveis às aplicações como memória, CPU e escalonamento para ver se possuem mecanismos diferenciados que possam dar suporte de QoS a aplicações multimídia distribuídas. Foram estudadas as plataformas Unix, através do Linux e Windows, através do Windows 2000.

6.1. Windows 2000

6.1.1. Alocação de Memória

A alocação de memória no Windows 2000 se dá de diversas maneiras: memória virtual, arquivos mapeados em memória, área de alocação dinâmica, pilhas e áreas locais de fluxos de execução.

As aplicações podem, primeiro, reservar espaços de endereço para depois executar o armazenamento nesses espaços de endereço. A reserva de memória é simplesmente uma maneira que uma *thread* tem de reservar um espaço de tamanho fixo de um endereço virtual para uso futuro chamado de página de memória.

A página de memória no espaço de endereço de um processo é livre, reservada ou comprometida. As páginas comprometidas também são privadas (e assim inacessíveis a qualquer outro processo) ou mapeadas em uma visão de uma sessão (que pode ou não pode ser mapeada por outro processo). A sessão representa um bloco de memória que dois ou mais processos podem compartilhar. Uma sessão pode ser papeada para um arquivo de pagina ou para um arquivo no disco.

O Windows 2000 nos permite descomprometer ou liberar espaços de endereço com funções específicas para isso onde os espaços reservados ficam livres para serem utilizados por qualquer outro processo [Silberchatz, 00].

6.1.2. Gerenciamento de Memória

O Windows 2000 pode limitar a quantidade de espaço no arquivo de paginação que um processo pode consumir, impondo um limite para a memória alocada a ele e ainda pode liberar a memória utilizada por um processo que não a esteja mais usando. Isto faz com que se tenha uma melhor distribuição dos recursos do sistema.

O sistema pode controlar uma sessão de memória compartilhada determinando o seu tamanho máximo que pode ser limitado. Ela pode ser copiada em um espaço em disco, no arquivo de paginação do sistema ou em um arquivo comum chamado arquivo mapeado em memória.

Para gerenciar os recursos existe ainda o administrador de E/S que é responsável pelos sistemas de arquivo, memórias cache, controladores de dispositivos e rotinas de controles de dispositivos de rede. Ele administra o uso de áreas de armazenamento para operações de E/S e trabalha conjuntamente com o administrador de memória virtual em operações de E/S em arquivos mapeados em memória e controla a administração de memórias cache para todo o sistema de E/S. Oferece suporte às operações síncronas e assíncronas, gera interrupções dependentes de tempo para rotinas de controle de dispositivos e ainda fornece mecanismos para que uma rotina de controle faça uma chamada para outra rotina.

Em muitos sistemas operacionais, o uso de memórias cache é controlado pelo sistema de arquivos, mas o Windows 2000 fornece um mecanismo de controle centralizado do uso desse tipo de memória. O administrador da memória cache oferece serviços para todos os componentes sob controle do administrador de E/S, trabalhando em conjunto com o administrador de memória virtual.

6.1.3. Escalonamento

Segundo [Carissimi, 01], os processos no Windows 2000 são implementados como objetos e são acessados usando serviços de objetos. O gerenciador de processos não mantém nenhuma relação pai e filho ou qualquer outro tipo de relação entre os processos por ele criado. Possui código executável e espaço de endereçamento privado e ainda

recursos do sistema, como semáforos, portas e arquivos. Conta também com pelo menos uma linha de execução, chamada *thread*.

A *thread* é a parte do processo que o *kernel* do sistema operacional escala para execução, sem ela o programa não executa. O Windows 2000 divide os processos em *threads* (unidades executáveis) que permitem que um único processo execute, ao mesmo tempo, partes diferentes de seu código e obtenha melhor utilização do processador. A *thread* é uma entidade executável que roda no espaço de endereçamento de um processo, usando recursos alocados ao processo.

O *kernel* do Windows 2000 é responsável por registrar as *threads* que estão prontas para serem executadas e selecionar a ordem em que elas serão executadas, ou seja, efetuar o escalonamento das *threads*. Quando for necessário, o *kernel* seleciona a próxima *thread* a ser executada e realiza uma troca de contexto para essa nova *thread*. O módulo do *kernel* que executa esse serviço é conhecido como *dispatcher* (despachante).

A tarefa do *dispatcher* é de garantir que de todas as *threads* que estão esperando para serem executadas, os processadores sempre executem as mais apropriadas. Quando o estado de uma *thread* é alterado, o *dispatcher* deve examinar a lista de *threads* em espera e realiza uma troca de contexto para uma nova *thread* se houver necessidade de alteração.

O *kernel* trabalha com uma versão reduzida do objeto de *thread*, que se chama *kernel thread object*. Um objeto desse tipo está contido dentro de um objeto de *thread* executivo e representa apenas as informações que de que o *kernel* precisa para a execução da *thread*. Da mesma forma, o *kernel* implementa uma versão mínima de um objeto processo, chamado de *kernel process object*. O *kernel process object* mantém um ponteiro para uma lista de *kernel thread objects*. O *kernel* também aponta para o diretório da tabela de página de processos, o tempo total de execução das *threads* de um processo e a prioridade padrão de escalonamento do processo.

O *kernel thread object* é mais complicado que o *kernel process object.* Ele contém informações como o tempo de execução de uma *thread*, a prioridade padrão de escalonamento da *thread* e a prioridade atual (pode ser diferente da padrão). Esse objeto também é responsável pelo armazenamento do estado em que se encontra a *thread*.

6.1.3.1. Estados de uma *thread*

O ciclo de vida de uma *thread* começa quando uma nova *thread* é criada por um programa. O gerenciador de processos aloca espaço para um objeto de *thread* e chama o *kernel* para inicializar o objeto de linha *kernel* contido dentro dele. Depois de inicializada, a *thread* passa pelos seguintes estados:

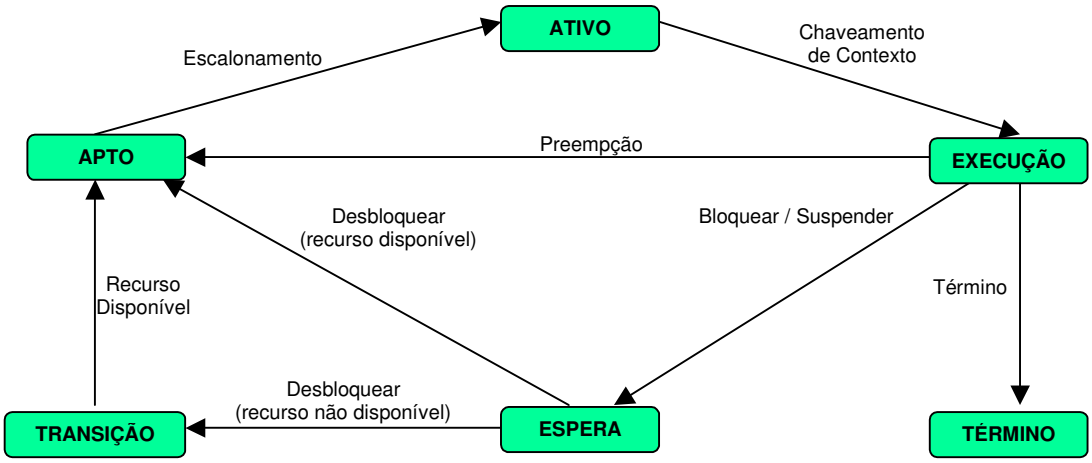

Figura 6 – Estados de uma *thread*

- *Apto (ready)*: corresponde ao estado no qual se encontram as *threads* aptas a executar, ou seja, as *threads* que o escalonador considera para selecionar a próxima a ser executada. Uma vez selecionada, a *thread* passa ao estado ativo (*standby*);

- *Ativa (standby)*: estado intermediário no qual a *thread* selecionada pelo escalonador espera pelo chaveamento de contexto para entrar efetivamente em execução. No sistema existe, por processador, apenas uma *thread* nesse estado;

- *Execução (running):* estado que assume uma *thread* quando está ocupando o processador. Uma *thread* em *running* executa até que ela seja preemptada por uma *thread* de mais alta prioridade, esgote a sua fatia de tempo, realize uma operação bloqueante, ou termine. Nos dois primeiros casos, o descritor da *thread* é reinserido na lista de aptos (*ready*);

- *Espera (waiting)*: uma *thread* passa a esse estado sempre que for bloqueada pela espera da ocorrência de um evento, quando realizar uma primitiva de sincronização ou quando o subsistema ordena a suspensão da *thread*. Quando a condição de espera é satisfeita, a *thread* é inserida na lista de aptos;

- *Transição (transition)*: corresponde ao estado em que uma *thread* está apta a ser executada, porém os recursos de sistema necessários a sua execução, ainda não estão disponíveis. Quando esses recursos estão disponibilizados, a *thread* passa ao estado apto;

- *Término (terminated)*: estado que uma *thread* assume quando atinge seu final ou é terminada por uma outra *thread*, ou ainda quando o processo a que está associada termina.

6.1.3.2. Prioridades de Escalonamento

O *dispatcher* do *kernel* usa um esquema de prioridades para determinar a ordem em que são executadas as *threads*. As de prioridade mais alta são escalonadas primeiro.O *kernel* interrompe a execução de uma *thread* caso uma *thread* com prioridade mais alta fique pronta pra executar (preempção).

Uma *thread* recebe a prioridade do processo em que foi criada. Quando um subsistema de ambiente cria um processo, o processo recebe uma prioridade padrão. Essa prioridade é então passada as *threads* criadas pelo processo e, com essa prioridade, a *thread* começa a executar. Com o tempo esse valor pode ser alterado.

O *kernel* mantém uma estrutura de dados conhecida como *database dispatcher*. Essa base de dados é usada pelo *dispatcher* para tomar as decisões de escalonamento. Essa base de dados contém uma lista com as *threads* que estão em execução e uma lista com as que estão em espera. A estrutura mais importante é o *dispatcher reading queue*. Esta lista contém uma série de outras listas, uma para cada uma das prioridades de escalonamento, com os processos prontos para serem executados.

O Windows 2000 suporta 32 níveis de prioridade dividido em duas classes, tempo real e prioridade variável. As *threads* de tempo real, cujas prioridades variam entre 16 e 31 são *threads* de alta prioridade, onde o tempo é o elemento crucial.

Quando o *dispatcher* reescalona um processador, ele recomeça na fila de prioridade mais alta e vem descendo até encontrar uma *thread*. A maioria das *threads* de sistema estão na classe de prioridades variável. Esse grupo recebe esse nome porque as *threads* que ele contém tem sua prioridade ajustada durante a execução, visando a otimização do tempo de resposta.

O Windows 2000 é um sistema multitarefa preemptivo, o *dispatcher* interrompe a execução de uma *thread* assim que ela esgota sua fatia de tempo no processador. Se a *thread* interrompida for de prioridade variável, sua prioridade é diminuída.

Por outro lado o *dispatcher* aumenta a prioridade de uma *thread* depois que ela é liberada de uma operação de espera. O código fora do *kernel* normalmente determina o tamanho do impulso de prioridade de uma *thread*. Uma *thread* que está numa entrada do teclado recebe um aumento de prioridade maior do que um que está aguardando a conclusão de um E/O de disco. No geral, as *threads* interativas tendem a rodar com prioridade variável alta, as *threads* vinculadas a operação de E/O em prioridade média e as *threads* vinculadas a cálculos em prioridade baixa.

6.1.3.3. Troca de Contexto

Depois que uma *thread* esgota sua fatia de tempo de execução, o *kernel* passa a executar e reescalona o processador. Mas não é somente o esgotamento da fatia de tempo que faz uma *thread* ser interrompida. Ela também pode ser interrompida por outros eventos, como a mudança de prioridade pelo *dispatcher*.

No reescalonamento de *threads*, o *kernel* usa o *dispatcher database* para determinar que processadores estão ocupados, quais estão ociosos e qual a prioridade da *thread* que cada um está processando.

O contexto de uma *thread* e o procedimento para troca de contexto varia dependendo da arquitetura de cada processador. Normalmente são salvos os contadores de

programa, o registro de status do processador, valores em registradores, ponteiros para pilhas do usuário e *kernel* e ainda ponteiro para espaço de endereçamento que a *thread* está usando.

Para efetuar a troca de contexto, o *kernel* salva estas informações colocando-as na pilha do modo *kernel* da *thread* atual e atualizando o ponteiro da pilha. O contexto da nova *thread* é carregado e o controle vai para o contador de programa da nova *thread*.

Uma *thread* que esteja executando código do executivo do Windows 2000 pode ser interrompida por uma *thread* de prioridade mais alta, o que não acontece com uma *thread* que esteja executando código *kernel*. Quando o *kernel* entra em execução, ele roda com prioridade mais alta que qualquer outra *thread* do sistema. Apesar disso, o *kernel* pode ser bloqueado por interrupções de nível mais alto.

6.2. Linux

6.2.1. Alocação de Memória

O administrador de memória física principal no Linux é o alocador de páginas onde tem como responsabilidade alocar e liberar páginas físicas, sendo capaz de alocar grupos de páginas contíguas.

As alocações de memória no núcleo do Linux ocorrem estaticamente, por rotinas de controle que reservam uma área contígua de memória no momento da carga do sistema, ou dinamicamente, pelo controlador de páginas. Entretanto as funções do núcleo não precisam usar o alocador de páginas para reservar memória, pois existem vários outros subsistemas de gerenciamento de memória especializados, que usam o controlador de páginas subjacente para gerenciar seu próprio espaço de memória. Os subsistemas mais importantes são o sistema de memória virtual, o sistema de alocação de áreas de memória de tamanho variável e o sistema de alocação de espaço nas duas memórias cache (cache temporária e cache de páginas).

De acordo com [Silberchatz, 00], o Linux possui também um sistema de memória virtual que é responsável pelo uso do espaço de endereçamento de cada processo. Esse sistema aloca espaço de memória virtual sob demanda e gerencia a transferência de

páginas entre o disco e a memória, quando necessário. O administrador de memória virtual usa duas visões do espaço de endereçamento de um processo: como um conjunto de regiões separadas e como um conjunto de páginas.

Uma região de memória virtual é também pela forma como são tratadas as operações de escrita sobre essa região. O mapeamento de uma região no espaço de endereçamento de um processo pode ser privado ou compartilhado. Se um processo escreve em uma região privada, o controlador de páginas detecta que é necessária uma *cópia-em-escrita*, para manter essas atualizações locais ao processo. Por outro lado, uma operação de escrita sobre uma região compartilhada resulta na atualização da cópia dessa região mantida na memória, de modo que essa atualização seja imediatamente visível a todos os processos que usam essa região.

6.2.2. Gerenciamento de Memória

Tanto o controlador de páginas como o sistema de alocação de áreas de memória de tamanho variável não podem ser interrompidos. Uma rotina que queira alocar uma área de memória informa a prioridade da sua requisição à rotina de alocação. Rotinas de interrupção utilizam uma prioridade atômica que garante que a requisição seja satisfeita ou que falhe imediatamente, caso não exista mais memória disponível. Em contraposição, para a requisição de memória de um processo comum de usuário, uma área de memória livre é procurada, sendo o processo bloqueado até que uma área de memória se torne disponível. A prioridade de alocação também pode ser usada para especificar a requisição de memória de acesso direto (DMA).

Outros subsistemas de memória que realizam um gerenciamento próprio de blocos de memória física são fortemente relacionados entre si. Esses sistemas gerenciam o uso da memória cache de áreas de armazenamento temporário, da memória cache de páginas e da memória virtual.

6.2.3. Escalonamento

O sistema operacional Linux possui três políticas de escalonamento diferentes: uma para processos “normais” (preemptivo) e duas para processos de tempo real (FIFO e *Round Robin*).

O escalonador preemptivo se utiliza de prioridades da seguinte forma: cada processo tem a sua prioridade (créditos iniciais). O escalonador escolhe o processo com o maior número de créditos e o executa até que fique com zero créditos (cada crédito corresponde a um tempo de execução) ou quando ocorre uma espera por eventos ou uma interrupção. São executados os processos ativos até que todos os processos ativos tenham zero créditos. Então, todos os processos (ativos e inativos) são creditados novamente da seguinte forma: recebem a sua prioridade acrescida da quantidade de créditos que possuíam dividido por dois (créditos = créditos / 2 + prioridade).

A política de escalonamento FIFO, também conhecida como FCFS (*First Come First Served*), utiliza uma fila de espera em que os processos que requisitam a CPU são colocados no final da mesma fila no momento em que a requisição é feita. Assim, os processos que requisitam a CPU antes são executados em primeiro lugar. Quando um processo que estava sendo executado pelo processador finaliza, ele é descartado e a CPU executa o processo a seguir da fila. Essa política de escalonamento é muito simples de ser implementada, porém o tempo de resposta dessa política é alto.

O algoritmo de escalonamento *Round Robin* utiliza uma fila circular em que os novos processos são colocados no final da fila, assim como ocorre na política FIFO. Porém, nessa política, cada processo possui uma pequena unidade de tempo usualmente chamada de quantum de tempo. Assim que cada processo executa um quantum de tempo, esse processo é interrompido e é colocado no final da fila, ocasionando uma troca de contexto, pois a CPU irá executar um quantum de tempo de um outro processo que esteja na fila. O tamanho do quantum, porém, não pode ser muito pequeno, pois isso ocasionará muitas trocas de contexto, retardando a execução de cada um dos processos, e também não pode ser muito grande, pois isso resultaria em um tempo de resposta muito alto, podendo até mesmo vir a se transformar na política FIFO.

Tradicionalmente, os processos são divididos em três grandes classes: processos interativos, processos *batch* e processos tempo real. Em cada classe, os processos podem ser ainda subdivididos em *I/O bound* ou *CPU bound* de acordo com a proporção de tempo que ficam esperando por operações de entrada e saída ou utilizando o processador. O escalonador do Linux não distingue processos interativos de processos *batch*, diferenciando-os apenas dos processos tempo real. Como todos os outros escalonadores

UNIX, o escalonador Linux privilegia os processos *I/O bound* em relação aos *CPU bound* de forma a oferecer um melhor tempo de resposta às aplicações interativas.

O escalonador do Linux é baseado em *time-sharing*, ou seja, o tempo do processador é dividido em fatias de tempo (*quantum*) as quais são alocadas aos processos. Se, durante a execução de um processo, o *quantum* é esgotado, um novo processo é selecionado para execução, provocando então uma troca de contexto. Esse procedimento é completamente transparente ao processo e baseia-se em interrupções de tempo. Esse comportamento confere ao Linux um escalonamento do tipo preemptivo.

O algoritmo de escalonamento do Linux divide o tempo de processamento em épocas (*epochs*). Cada processo, no momento de sua criação, recebe um *quantum* calculado no início de uma época. Diferentes processos podem possuir diferentes valores de *quantum*. O valor do *quantum* corresponde à duração da época, e essa, por sua vez, é um múltiplo de 10 ms inferior a 100 ms. Outra característica do escalonador Linux é a existência de prioridades dinâmicas. O escalonador do Linux monitora o comportamento de um processo e ajusta dinamicamente sua prioridade, visando equalizar o uso do processador entre os processos. Processos que recentemente ocuparam o processador durante um período de tempo considerado “longo” têm sua prioridade reduzida. De forma análoga, aqueles que estão há muito tempo sem executar recebem um aumento na sua prioridade, sendo então beneficiados em novas operações de escalonamento.

Na realidade, o sistema Linux trabalha com dois tipos de prioridades: estática e dinâmica. As prioridades estáticas são utilizadas exclusivamente por processos de tempo real correspondendo a valores na faixa de 1-99. Nesse caso, a prioridade do processo tempo real é definida pelo usuário e não é modificada pelo escalonador. Somente usuários com privilégios especiais têm a capacidade de criar e definir processos tempo real. O esquema de prioridades dinâmicas é aplicado aos processos interativos e *batch*. Aqui, a prioridade é calculada, considerando-se a prioridade base do processo e a quantidade de tempo restante em seu *quantum*.

O escalonador do Linux executa os processos de prioridade dinâmica apenas quando não há processos de tempo real. Em outros termos, os processos de prioridade estática recebem uma prioridade maior que os processos de prioridade dinâmica. Para

selecionar um processo para execução, o escalonador do Linux prevê três políticas diferentes:

- *FIFO:* Essa política é valida apenas para os processos de tempo real. Na criação, o descritor do processo é inserido no final da fila correspondente à sua prioridade. Nessa política, quando um processo é alocado ao processador, ele executa até que uma de três situações ocorra: (1) um processo de tempo real de prioridade superior torna-se apto a executar; (2) o processo libera espontaneamente o processador para processos de prioridade igual à sua; (3) o processo termina, ou bloqueia-se, em uma operação de entrada e saída ou de sincronização;

- *Round Robin*: Na criação, o descritor do processo é inserido no final da fila correspondente à sua prioridade. Quando um processo é alocado ao processador, ele executa até que uma de quatro situações ocorra: (1) seu período de execução (*quantum*) tenha se esgotado nesse caso o processo é inserido no final de sua fila de prioridade; (2) um processo de prioridade superior torna-se apto a executar; (3) o processo libera espontaneamente o processador para processos de prioridade igual a sua; (4) o processo termina, ou bloqueia-se, em uma operação de entrada e saída ou de sincronização. Essa política também só é válida para processos de tempo real;

- *Filas Múltimplas:* Corresponde a um esquema de filas multinível de prioridades dinâmicas com *timesharing*. Os processo interativos e *batch* recaem nessa categoria. Nessa política existem várias filas de processos e cada processo é colocado em uma determinada fila, sendo que cada fila possui sua própria política de escalonamento e é implementada uma política de escalonamento entre as filas.

A criação de processos em Linux, como em todos os sistemas UNIX, é baseada em uma operação do tipo *fork-exec*, ou seja, um processo cria uma cópia sua (*fork*) e em seguida substitui o seu código por um outro (*exec*). No momento da criação, o processo pai (o que fez o *fork*) cede metade de seu *quantum* restante ao processo filho. Esse procedimento é, na verdade, uma espécie de proteção que o sistema faz para evitar que um

usuário, a partir de um processo pai, crie um processo filho que execute o mesmo código do pai. Sem essa proteção, a cada criação o filho receberia um novo *quantum* integral. Da mesma forma, o núcleo Linux previne-se contra o fato de um mesmo usuário lançar vários processos em *background*, ou executar diferentes sessões *shell*.

O escalonador do Linux é executado a partir de duas formas diferentes. A primeira é a forma direta através de uma chamada explícita à rotina que implementa o escalonador. Essa é a maneira utilizada pelo núcleo do Linux quando, por exemplo, detecta que um processo deverá ser bloqueado em decorrência de uma operação de entrada e saída ou de sincronização. A segunda forma, denominada de *lazy*, também é conseqüência do procedimento de escalonamento, ocorrendo tipicamente em uma de três situações.

A primeira dessas situações é a rotina de tratamento de interrupção de tempo que atualiza os temporizadores e realiza a contabilização de tempo por processo. Essa rotina, ao detectar que um processo esgotou seu *quantum* de execução aciona o escalonador para que seja efetuada uma troca de processo. A segunda situação ocorre quando um processo de mais alta prioridade é desbloqueado pela ocorrência do evento que esperava. A parte do código que efetua o desbloqueio, isto é, trata os eventos de sincronização e de entrada e saída, consulta a prioridade do processo atualmente em execução e compara-a com a do processo que será desbloqueado. Se o processo a ser desbloqueado possuir uma prioridade mais alta, o escalonador é acionado, e ocorre uma troca de processo. A terceira, e última forma de execução *lazy,* é quando um processo explicitamente invoca o escalonador através de uma chamada de sistema do tipo *yield*. Essa chamada de sistema permite a um processo “passar sua vez de execução” a outro processo, e, para isso, parece claro, é necessário executar o escalonador.

7. Conclusão

Qualidade de Serviço (QoS) é uma especificação qualitativa e quantitativa dos requisitos de uma aplicação que um sistema multimídia deve satisfazer a fim de obter a qualidade desejada [Lu, 96]. QoS é um requisito fundamental para diversas aplicações e envolve atividades como: especificação, negociação e gerenciamento dos recursos disponíveis.

Como foi visto neste trabalho, as apresentações multimídia, especialmente as de mídia contínua, como áudio e vídeo, necessitam que um certo conjunto de requisitos seja definido para que possam ser executadas com qualidade.

Dentre os requisitos estudados está um suporte a tempo real, pois em muitas aplicações dados são transmitidos, processados e apresentados com uma taxa fixa. Devem ser utilizadas redes de alta velocidade e ainda compactar os dados no servidor antes da transmissão e descompactar na máquina cliente devido a grande volume de dados de tais aplicações. E ainda como as aplicações multimídia usam simultaneamente vários tipos de mídia devem ser mantidas as relações temporais e espaciais entre essas mídias.

Com relação à reserva de recursos em nível de rede foi visto que o protocolo RSVP é de fundamental importância para que se tenha uma apresentação com qualidade, já que este foi projetado para aumentar o suporte para aplicações tempo-real em redes IP. Como foi detalhado no decorrer deste trabalho, o RSVP não executa a reserva do recurso em si, ele é apenas o veículo para transferir informações acerca dos requisitos de recursos e é usado para negociar os valores de QoS que o usuário deseja para suas aplicações.

Quanto ao sistema operacional constatou-se que é um componente de vital importância em um sistema multimídia, principalmente da forma como controla a especificação e requisição de alocação para os recursos que são requeridos ao executar tarefas com a qualidade de serviço solicitada e ainda com os mecanismos de gerenciamento de memória e escalonamento onde assegura que um compartilhamento suficiente dos recursos esteja disponível todo o tempo.

Mais especificamente aos algoritmos de escalonamento, foi visto que um algoritmo viável para aplicações multimídia deve ser baseado nos requisitos de QoS da aplicação, mesmo em situações de sobrecarga de processamento. Ainda deve utilizar um tempo mínimo de processador para as decisões de escalonamento, minimizando seu *overhead* e assegurar, sempre que possível, que todas as tarefas que manipulam mídias contínuas sejam atendidas prioritariamente e terminem sua execução dentro de suas respectivas *deadlines*, mas sem deixar de atender as tarefas que manipulam mídias discretas.

Nos sistemas operacionais estudados tanto o Windows 2000 como o Linux não possuem mecanismos diferenciados para dar suporte a aplicações multimídia distribuídas, pois verificou-se que nenhum atende os requisitos mínimos para esse tipo de aplicações. Isso ficou constatado no estudo feito com os principais recursos do SO como alocação e gerenciamento de memória, alocação e gerenciamento da CPU e, principalmente, escalonamento.

8. Referências Bibliográficas

As referências abaixo relacionadas são apresentadas ao longo dos capítulos do presente trabalho:

[Bollella, 95] Bollella, G., Jeffay, K. *Support co-resident operating system*. Jun 1995.

[Braden, 96] Braden, B. *Resource Reservation Protocol (RSVP). Version 1 Functional Specification.* Internet Draft, March 1996.

[Campbell, 97] Campbell, A.T., Aurrecoechea, C., Hauw, L. *A Survey of QoS Architectures, Multimedia Systems Journal , Special Issue on QoS Architecture*, 1997.

[Campbell, 97] Campbell, A.T., Coulson, G., Hutchison. *A Quality of Service Architecture*. April, 1997.

[Carissimi, 01] A. Carissimi, R. Oliveira, S. Toscani. *Sistemas Operacionais*. Editora Sagra-Luzzato, 2001.

[Claypool, 98] M. Claypool and J. Riedl. *End-to-End Quality in Multimedia Applications*. Handbook on Multimedia Computing, CRC Press, Boca Roton, FL, 1998.

[De Castro] De Castro, Maria Clicia Stelling. *Sistemas Operacionais*. Apostila da disciplina de Sistemas Operacionais, Universidade Federal de juiz de Fora.

[Diot, 95] Diot, Ch. *Adaptative Applications and QoS Guarantees*. Proc. IEEE International Conference on Multimedia and Networking. IEEE Computer Society Press. Japan, 1995.

[Fluckiger, 95] Fluckiger, François. *Understanding Networked Multimedia: Applications and Technology*. Prentice Hall International (UK) Limited, 1995.

[Furht, 94] Furht, Borko. *Multimidia Systems: An Overview*. IEEE Multimidia Spring, 1994.

[Gecsei 97] Gecsei, J. *Adaptation in Distributed Multimedia Systems.* IEEE Multimedia, Vol. 4, No. 2, April-June 1997.

[Guedes, 97] Guedes, L. A., Cardozo, E. *Especificação de um Protocolo para Negociação de Qualidade de Serviço em Sistemas Multimídia.* Unicamp e Universidade Federal do Pará, 1997.

[Herrtwich , 91] Herrtwich, R. G. *The Role of Performance, Scheduling, and Resource Reservation in Multimedia Systems*. Operating Systems of the 90s and Beyond, A Karshmer and J. Nehmer, eds. Springer-Verlag, Berlin, 1991.

[Jeffay, 95] K. Jeffay and D. Bennett. *A Rate-Based Execution Abstraction for Multimedia Computing*. In Proc. Of 5th Intl. Workshop on Network and Operating System Support for Digital Audio and Video, 1995.

[Kumar, 95] Kumar, V. *MBone: Interactive Multimedia on the Internet*. Indianopolis: New Riders Publishing, 1995.

[Little, 96] Little, T. D. C., Luque-Perez, M. J. *A temporal reference framework for multimedia synchronization*. IEEE J. Select. Areas commun., vol. 14, Jan 1996.

[Lakshman, 97] Lakshman, K.: *AQUA: An Adaptive Quality of Service Architecture for Distributed Multimedia Applications.* University of Kentucky, 1997.

[Liao, 97] Liao, W., Li, V. O. K. *Distributed Multimedia Systems*. Proceedings of IEEE, New York, vol. 85, July 1997.

[Lu, 96] Lu, Guojun. *Communication and Computing for Distributed Multimedia Systems*. Artech House Inc., 1996.

[Mercer, 94] Mercer, C. W., Savage, S., Tokuda, H. *Processor Capacity Reserves for Multimídia Operating System*. May 1994.

[Nahrstedt, 95] Nahrstedt, K., Smith, J. M. *The QoS Broker*. IEEE Multimidia, Vol 2, Spring 1995.

[Nieh, 95] J. Nieh and M. Lam. *SMART: A Processor Scheduler for Multimedia Applications*. In Proc. of SOSP 15, December 1995.

[Pawan, 96] X. G. Pawan Goyal and H. M. Vin. *A Hierarchical CPU Scheduler for Multimedia Operating Systems*. In 2nd Symposium on Operating Systems Design and Implementation. USENIX, 1996.

[Peterson, 85] Peterson, J.L., Silberschatz, A. *Operating System Concepts*. Addison-Wesley, 1985.

[Silberchatz, 00] Silberchatz, A., Galvin P. *Operating System Concepts*. Addison-Wesley, 2000.

[Stachli, 96] Stachli, R. A. *Quality of Service Specification of Resource Management in Multimidia Systems*. Jan 1996.

[Sommer, 96] Sommer, S., Potter, J. *Operating Systems Extensions for Dynamic Real-time Applications*. Dec 1996.

[Tanembaum, 96] Tanembaum, A. *Computer Networks* - 3a. edição - 1996.

[Vogel, 95] Vogel A., Kerhervé B., Bocchmann G., Gecsei J., *Distributed Multimidia and QOS: A Survey*. IEEE Multimedia Summer 1995.

[Zhang, 94] Zhang, L. et al. *Resource Reservation Protocol (RSVP)*. Internet Draft, Mar 1994.

[Yau, 96] D. Yau and S. Lam. An Architecture Towards Efficient *OS Support for Distributed Multimedia*. In Proc. of IS&T/SPIE Multimedia Computing and Networking Conference, January 1996.

[Watson, 96] Watson, Ana, Sasse, M. *Multimedia Conferencing via Multicast: Determining the Quality of Service Required by the End User.* Technical Report, University College Science, 1996.

[West, 99] R. West, K. Schwan, and C. Poellabauer. *Scalable Scheduling Support for Loss and Delay Constrained Media Streams*. In Proc. of the 5th Real-Time Technology and Applications Symposium, Vancouver, Canadá, 1999.

[Willrich, 00] Willrich, Roberto. *Sistemas Multimídia Distribuídos*. Apostila da disciplina de Sistemas Multimídia Distribuídos, UFSC, Mar 2000.

[Wolf, 97] Wolf, L. C., Griwodz, C., Steirmetz, R. *Multimedia Communications*. Proceedings of IEEE, New York, vol. 85, Dec 1997.