# 2
# QoS em Sistemas Operacionais

Este Capítulo tem o objetivo de evidenciar as deficiências encontradas nos subsistemas de rede e de escalonamento de processos de sistemas operacionais de propósito geral (GPOS[3]), no que se refere ao oferecimento de serviços com QoS, e mostrar as soluções descritas em vários trabalhos relacionados. Entre esses trabalhos, foram encontrados três tipos principais de estudo: estudos isolados sobre os subsistemas citados, independentes do sistema operacional; estudos para o desenvolvimento de novos sistemas operacionais; e estudos para a extensão dos sistemas operacionais existentes, principalmente os GPOS.

Identificando os principais mecanismos de provisão de QoS descritos por esses trabalhos, podem ser determinados seus aspectos em comum e, assim, fundamentar a construção dos frameworks propostos. A terminologia utilizada para a descrição das funcionalidades em questão é a mesma definida em (Gomes, 1999).

Primeiramente, serão apresentadas as fases que compõem a provisão de QoS no contexto de sistemas operacionais, descrevendo quais as funções envolvidas em cada uma delas. Após a análise sobre os GPOS e trabalhos relacionados, será feita uma modelagem genérica do funcionamento do sistema, onde estarão representados os subsistemas de processamento e comunicação e seus recursos, que participam do oferecimento de serviços às aplicações. Será identificado um importante relacionamento entre esses subsistemas, que deve ser considerado na construção dos frameworks propostos no Capítulo 3.

[3] GPOS – Sigla para General Purpose Operating System – será utilizada para identificar os sistemas operacionais de propósito geral, tanto no singular quanto no plural. Os GPOS são os sistemas encontrados comercialmente com certa popularidade e que não se dedicam a tarefas e aplicações específicas.

## 2.1
## Fases da Provisão de QoS

A provisão de QoS requer que o fornecedor do serviço, no caso o sistema operacional, ofereça certos mecanismos que serão utilizados em fases bem definidas do ciclo de vida de um serviço. Tais fases serão descritas a seguir, destacando as funções e estruturas básicas necessárias em cada uma delas.

- Iniciação do Provedor de Serviços
- Solicitação de Serviços
- Estabelecimento de Contratos de Serviço
- Manutenção de Contratos de Serviço
- Encerramento de Contratos de Serviço



### 2.1.1
### Iniciação do Provedor de Serviços

A iniciação do provedor de serviços compreende a *identificação dos recursos disponíveis* para o fornecimento dos serviços e a *definição do estado interno inicial* do provedor. É desejável que a associação dos recursos aos serviços seja feita dinamicamente, no momento da solicitação, tanto para que os recursos possam ser alocados de forma mais eficiente, quanto para que o sistema ofereça uma maior flexibilidade aos usuários. Associações estáticas, no entanto, podem ser feitas em tempo de projeto e ativadas nesta fase de iniciação.

Em sistemas operacionais, a iniciação pode ser feita no momento de carga do sistema, quando o *kernel* conhecerá os recursos disponíveis e poderão ser determinadas as suas estruturas iniciais para alocação. Dependendo da *adaptabilidade* do sistema, o serviço poderá ser reiniciado a partir de um novo estado interno[4] inicial, sem que o sistema operacional tenha que ser recarregado. Em um nível de adaptabilidade ainda maior estariam os sistemas que podem *incluir novos serviços em tempo de operação*, agregando um novo conjunto de

[4] O estado interno de um sistema compreende a situação de particionamento de cada um dos recursos gerenciados e os conjuntos de políticas de QoS para uso pelos serviços oferecidos.

*políticas de provisão de QoS* ao seu estado interno anterior. Dessa forma, o sistema operacional terá a capacidade de adotar, por exemplo, novas estratégias de escalonamento e de admissão de recursos.

### 2.1.2 Solicitação de Serviços

Completada a fase de iniciação, o provedor se encontra pronto para receber as solicitações de serviços dos usuários. Num sistema operacional, a solicitação pode partir diretamente da aplicação ou pode ter sido repassada por um outro nível de abstração, através, por exemplo, de um protocolo de negociação de QoS. A solicitação contém os parâmetros que correspondem à *caracterização do tráfego* para os fluxos[5] gerados pelo usuário e à *especificação da QoS* necessária.

Dependendo do *nível de abstração* no qual estão definidos os parâmetros da solicitação, o sistema deve possuir funções de *mapeamento* capazes de traduzi-los para valores relacionados diretamente com a capacidade de processamento ou comunicação dos recursos. Por exemplo, no nível da aplicação, a especificação da QoS de um sistema de vídeo sob demanda poderia envolver a taxa de atualização do vídeo e o tamanho de seu quadro. No nível do sistema, esses parâmetros devem ser mapeados para a taxa de pico de transmissão de bits através de um enlace de rede e para a taxa de instruções executadas por uma CPU para tratar o vídeo.



### 2.1.3 Estabelecimento de Contratos de Serviço

Após a solicitação de um serviço, o provedor deve executar suas funções de *controle de admissão* para verificar a viabilidade da aceitação de um novo fluxo, de forma que a QoS solicitada seja garantida. As *estratégias de admissão* se baseiam em informações sobre as *reservas de recursos* já efetuadas ou em

[5] Neste trabalho, um fluxo pode representar uma seqüência de unidades de informação (e.g. pacotes de dados) transmitida entre processos (dispostos local ou remotamente) ou uma seqüência de instruções a ser executada pelo microprocessador de uma estação. Esta diferenciação será discutida na Seção 2.3.

medições da utilização real dos recursos para determinar se o novo serviço poderá ser aceito, dadas a especificação da QoS e a caracterização do tráfego gerado.

De acordo com a *categoria de serviço* solicitada, as estratégias de admissão podem se comportar de maneira conservadora ou agressiva. Estratégias de admissão conservadoras são aquelas que calculam a reserva dos recursos de forma menos eficiente, permitindo um número bem menor de fluxos coexistentes, mas com fortes garantias de *manutenção da QoS*. Já as estratégias agressivas avaliam as reservas de forma mais inteligente, admitindo um maior número de fluxos, mas correndo um certo risco de ocorrência de sobrecarga sobre o recurso em algum momento.

Se o controle de admissão concluir que a solicitação do serviço não pode ser atendida, o *contrato de serviço* não é estabelecido, podendo o usuário realizar uma requisição equivalente em um outro momento ou, imediatamente, especificar parâmetros mais relaxados para o serviço.

Por outro lado, se a admissão for bem sucedida, são acionados os métodos para a criação dos *recursos virtuais*[6], através da configuração dos escalonadores de recursos, dos classificadores e dos agentes de policiamento. Assim, implicitamente está estabelecido o contrato de serviço, que obriga o provedor a manter o nível de QoS solicitado ao longo do período de provisão e o usuário a submeter seu fluxo em conformidade com a caracterização informada.

As operações que compõem a fase de estabelecimento de contratos de serviços são providas por mecanismos de *negociação de QoS*. Esses mecanismos também são responsáveis pela *orquestração dos recursos*, que consiste na divisão de responsabilidade da negociação de QoS entre os subsistemas que integram o provedor de serviços ou entre os múltiplos recursos de um mesmo subsistema.

Particularmente, em sistemas operacionais ocorrem ambas situações de orquestração de recursos. Por exemplo, em aplicações multimídia distribuídas, o negociador de QoS de um sistema operacional deve delegar a negociação aos

[6] Um recurso virtual corresponde à parcela reservada do recurso para uso exclusivo do fluxo submetido pelo usuário. Este conceito será melhor discutido no Capítulo 3.

mecanismos específicos dos principais subsistemas envolvidos, no caso, os subsistemas de rede e de gerenciamento de processos. Por sua vez, o subsistema de rede pode abranger o gerenciamento de várias filas de pacotes presentes em diferentes níveis da pilha de protocolos de rede. O negociador de QoS do subsistema de rede deve, então, dividir sua responsabilidade entre negociadores que poderiam estar associados a cada um dos níveis de protocolo de rede, onde houver retenção de pacotes para a comunicação com o próximo nível da pilha.

### 2.1.4 Manutenção de Contratos de Serviço

Para manter os acordos estabelecidos pelos contratos de serviço, o provedor deve assegurar que os serviços sejam oferecidos conforme as especificações de QoS solicitadas e que os fluxos submetidos pelos usuários estejam conformes com relação aos perfis de tráfego fornecidos. O comportamento inadequado de ambas as partes frente ao contrato estabelecido estará sujeito a penalidades que vão de uma simples notificação ao usuário até a interrupção abrupta da provisão do serviço.

Em sistemas operacionais, os mecanismos necessários para a operação de um serviço com QoS compreendem as funções de *classificação dos fluxos* e *escalonamento de recursos*. A manutenção dos contratos envolve a *monitoração* e *sintonização de QoS*.

A classificação dos fluxos tem o objetivo de identificar a categoria de serviço a que pertence uma dada unidade de informação componente do fluxo e, conseqüentemente, o escalonador de recursos a ser utilizado. Um exemplo é o classificador de pacotes de rede, que identifica a categoria de serviço de um pacote analisando dados contidos no seu cabeçalho.

Os escalonadores de recursos gerenciam o compartilhamento do recurso entre os vários fluxos que necessitam utilizá-lo. Esse compartilhamento deve seguir as especificações de QoS de cada usuário, protegendo-os individualmente das sobrecargas que podem ser geradas por alguns deles. Para isso, é comum o

emprego de vários algoritmos de escalonamento sobre um mesmo recurso, dadas as diferentes necessidades de QoS impostas pelas aplicações.

Como forma de verificar a conformidade dos fluxos gerados pelo usuário e a existência de sobrecarga em certos recursos, o sistema utiliza a monitoração da QoS. O policiamento de fluxos é um tipo de monitoração que faz medições do tráfego gerado pelo usuário e pode tomar decisões que evitem a violação da QoS especificada. Por exemplo, o policiamento de um fluxo de pacotes de rede detecta quais desses pacotes estão além da caracterização de tráfego informada, podendo apenas marcá-los como não-conformes ou descartá-los definitivamente, evitando que sejam processados.

Já os mecanismos de monitoração do próprio sistema verificam a carga sobre os recursos e qual a real QoS oferecida a cada um dos fluxos. Situações de sobrecarga podem ocorrer devido a estratégias de reserva de recursos pouco conservadoras ou pela falha de algum componente do sistema. Caso seja realmente detectada uma violação do contrato estabelecido, o sistema pode tomar algumas decisões, como apenas alertar os usuários afetados, ou dar início a uma renegociação de QoS, ou, ainda, ativar mecanismos de sintonização de QoS.

A sintonização de QoS envolve uma nova orquestração dos recursos, quando podem ser utilizados subsistemas ou recursos alternativos (e.g. filas de interfaces de rede que podem ser uma alternativa para a comunicação desejada), apesar de serem pouco comuns em sistemas operacionais. Pode-se, também, redistribuir a carga entre os recursos, ajustando-se seus parâmetros de escalonamento.

### 2.1.5 Encerramento de contratos de serviços

Findado o interesse de uma das partes em utilizar/prover o serviço, ou ainda, expirado o tempo de duração especificado no contrato, dá-se início à fase de encerramento do contrato de serviço. Recebida a requisição de término, gerada pelo usuário ou automaticamente, o provedor deve liberar, em cada um dos

subsistemas envolvidos com o fornecimento do serviço, os recursos reservados e seus classificadores e policiadores.

## 2.2 Características relevantes de sistemas operacionais de uso geral

A principal função de um sistema operacional é o gerenciamento dos recursos computacionais de uma estação, disponibilizando-os para quaisquer aplicações que necessitem realizar a transferência de dados entre eles. Alguns exemplos desses recursos são: microprocessadores, memórias, dispositivos de armazenamento secundário, buffers de comunicação e interfaces de entrada e saída de dados.

Os GPOS são sistemas de tempo compartilhado, no qual várias aplicações disputam simultaneamente o uso dos recursos citados acima. A evolução desses sistemas é caracterizada, especialmente, pela otimização do tempo despendido no controle dessa concorrência. Por isso, os desenvolvedores vinham trabalhando, principalmente, na redução de complexidade dos algoritmos de escalonamento de recursos, na diminuição do tamanho do código do *kernel* e no uso otimizado das propriedades específicas da arquitetura de hardware, como instruções exclusivas de certos microprocessadores.

Porém, várias características dos sistemas operacionais de uso geral, incluindo a simplicidade com que é tratada a alocação de recursos, impossibilitam o oferecimento de serviços com QoS a aplicações que possuem fortes exigências de processamento e comunicação. Nesta Seção, serão discutidas as características dos GPOS que influenciam, impossibilitam, ou são ausentes, para a provisão de qualidade de serviço e suporte à adaptabilidade.

### 2.2.1 Arquitetura de sistemas operacionais

A maioria dos GPOS, tais como Linux e Windows, está organizada internamente em uma estrutura monolítica, ou seja, é formada por um grande núcleo que abrange todas as funções de controle do sistema. Funções como

escalonamento de processos, gerenciamento de memória, comunicação entre processos e pilha de protocolos estão todas elas implementadas no *kernel*. Historicamente, a escolha por essa arquitetura se deve à facilidade de comunicação entre os módulos internos, através de chamadas de função com passagem de parâmetros, dando grande eficiência ao sistema (Maxwell, 2000). Outra vantagem dos sistemas monolíticos é a forte associação com as funcionalidades específicas da plataforma de hardware, tirando um melhor proveito dos benefícios oferecidos por cada uma delas.

Por outro lado, a arquitetura monolítica apresenta grande dificuldade de substituição de módulos internos por implementações mais eficientes, para um dado tipo de serviço, em tempo de projeto ou em tempo de execução. A capacidade de inserção e alteração de parte do código do *kernel* durante o funcionamento de um sistema operacional integra uma das formas de suporte à *adaptabilidade* do sistema para a inclusão de novos serviços. Além disso, a portabilidade do sistema se torna uma operação extremamente complexa, já que a migração do código de um *kernel* monolítico para uma outra plataforma de hardware leva à modificação de grande parte do seu extenso código.

Os GPOS se caracterizam, ainda, pelo processamento não-preemptivo do kernel, ou seja, a execução das instruções que integram o núcleo do sistema não pode ser interrompida para que a CPU seja utilizada por um outro processo. Por exemplo, um processo de alta prioridade na fila de execução da CPU deve esperar que uma chamada de sistema, disparada anteriormente por um processo de baixa prioridade, seja executada até seu fim, para que ocorra a preempção. Essa situação é chamada *inversão de prioridade* e é agravada pelo grande número de instruções que podem compor a execução de uma chamada de sistema em um *kernel* monolítico. Na provisão de serviços com QoS, a ocorrência de inversão de prioridades deve ser reduzida ao máximo.

### 2.2.2 Escalonamento de processos

As aplicações em execução em uma estação são representadas no sistema operacional por estruturas chamadas processos. Depois de carregado, um processo

entra em uma fila de espera pelo uso da CPU, chamada de fila de prontos, contendo processos no estado executável (runnable). O controle de acesso dos vários processos em execução à CPU é feito pelo escalonador de processos, que arbitra os intervalos de tempo a que eles terão direito.

O escalonamento de processos em um sistema operacional está sujeito a vários tipos de aplicações, que possuem diferentes necessidades de processamento. Em (Goyal, 1996b) são descritas três categorias principais de aplicações que podem coexistir em um sistema multimídia, considerando os requisitos quanto ao tempo de uso da CPU:

- Aplicações em tempo real severo (hard real-time): São aquelas que requerem garantias determinísticas sobre o retardo característico de ambientes multitarefa. Ex.: Controle distribuído de processos industriais.
- Aplicações em tempo real suave (soft real-time): Requerem garantias estatísticas sobre os parâmetros de qualidade de serviço, tais como retardo máximo e vazão. O sistema operacional deve ser capaz de utilizar a CPU de forma eficiente, por meio de over-book[7] da largura de banda. Ex.: Vídeo sob demanda.
- Aplicações de melhor esforço (best-effort): São as aplicações convencionais que não necessitam de garantias de performance, requerendo apenas que a CPU seja alocada de forma que o tempo médio de resposta seja baixo e a vazão atingida seja alta. Ex.: Transferência remota de arquivos.

Os escalonadores de processos dos GPOS, contudo, são baseados em algoritmos de compartilhamento de tempo, como round robin (Tanenbaum, 1992), incapazes de oferecer quaisquer garantias sobre o tempo máximo de acesso à CPU (retardo máximo), por exemplo. Para a execução de processos em tempo real, o escalonador recorre ao uso de prioridades, o que pode levar à inanição aplicações de melhor esforço. Quando uma aplicação requer seu processamento em tempo

[7] O termo over-book, de origem inglesa, tem aqui o mesmo significado de quando é usado pelas companhias aéreas. Ele significa que as reservas para um vôo (CPU) contam com possíveis desistências (utilização da CPU não chega ao máximo), colocando mais passageiros (processos) na fila de espera do que realmente comportaria o avião.

real, a mais alta prioridade de execução é atribuída ao seu processo, que somente libera a CPU por vontade própria, ou para eventos do sistema com maior prioridade, ou durante a espera por uma operação de entrada ou saída. Esse esquema é injusto, já que apenas a aplicação em tempo real tem suas necessidades de processamento garantidas, enquanto pode ocorrer que outros processos não consigam a posse da CPU por um longo período de tempo.

Problemas com prioridades também são observados no funcionamento de subsistemas que utilizam interrupções para a transferência do processamento aos drivers de dispositivos ou a outras funções do *kernel*. Isso será visto com mais detalhes na descrição do subsistema de rede dos GPOS.

Dadas as diferentes necessidades das várias categorias de aplicações, nota-se que diferentes devem ser os algoritmos para escalonamento de processos. Aplicações em tempo real severo são bem atendidas por algoritmos de escalonamento baseados no tempo limite para execução, como Earliest Deadline First (EDF) e Rate Monotonic Algorithm (RMA) (Liu, 1973). Os processos da categoria tempo real suave devem ser escalonados por algoritmos que ofereçam certas garantias de QoS, mesmo na presença de sobrecarga, como o Start-time Fair Queuing (SFQ) (Goyal, 1996a). Já os algoritmos de compartilhamento de tempo, presentes nos GPOS, satisfazem as necessidades das aplicações de melhor esforço.

Os algoritmos de escalonamento de tempo real severo, como EDF e RMA, não são adequados para aplicações de tempo real suave, por não poderem admitir processos em over-book. Além disso, esses escalonadores exigem um conhecimento a priori do período e do tempo de execução, detalhes de difícil descrição para tais aplicações.

Assim, é necessário que o escalonamento de processos em sistemas operacionais permita a adaptação de escalonadores pelo sistema, conforme as necessidades das aplicações. Ao mesmo tempo, deve prover proteção entre as várias categorias de aplicação, facilitando a coexistência das políticas de escalonamento. Por exemplo, o overbooking da CPU para as aplicações de tempo real suave não deve violar as garantias dadas às aplicações de tempo real severo.

Da mesma forma, um comportamento anormal de aplicações de tempo real severo e suave não deve levar as aplicações de melhor esforço à inanição. Essa propriedade é também chamada de isolamento entre as categorias de escalonamento.

Nos GPOS, existem apenas as categorias de melhor esforço e, algumas vezes, de tempo real, que são regidas pelo mesmo algoritmo de escalonamento, com o uso de prioridades para os processos de tempo real. Dessa forma, os escalonadores dos GPOS não promovem um acesso justo à CPU para todos os processos em execução nem garante o isolamento entre as categorias.

### 2.2.3 Subsistema de rede

O subsistema de rede de sistemas operacionais é composto pelo conjunto de operações que manipulam o recebimento e envio de pacotes através de um enlace de rede. Para identificar as deficiências desse subsistema nos GPOS, é apresentada na Tabela 2.1 a seqüência de passos que compõem o processamento de rede sob o padrão 4.4BSD[8] (Wright, 1995). Na Figura 2.1 cada um dos passos está ilustrado ao longo da pilha de protocolos de rede.

Nota-se que o sistema pode ser analisado sob vários aspectos. Sob o foco da otimização de protocolos, os GPOS possuem um número elevado de leituras e cópias de dados ao longo da execução da pilha de protocolos. Tais operações devem ser evitadas por demandarem tempo de CPU para leitura e escrita e, ainda, espaço extra de memória para o armazenamento, no caso da cópia.

Com relação à provisão de QoS, estão listados a seguir os principais pontos críticos da arquitetura do subsistema de rede dos GPOS, alguns deles descritos em trabalhos como (Druschel, 1996), (Shi, 1998) e (Coulson, 1995):

1. Chamadas de sistema (1): Usadas pela aplicação tanto para a transmissão quanto para a recepção de dados, as chamadas de sistema

[8] O padrão BSD4.4 é utilizado em sistemas como UNIX e Linux. São pequenas as diferenças entre o padrão 4.4BSD e o subsistema de rede da família Windows (Microsoft, 1996).

constituem uma das formas de comunicação entre os níveis de usuário e de kernel. O tempo de CPU gasto pelo kernel é computado para o processo chamador, o que não provoca anomalias no escalonamento (o tempo de consumo da CPU é levado em consideração na decisão de escalonamento). O problema está no fato de que os GPOS não possibilitam a preempção do processamento do kernel, obrigando que outras operações somente sejam executadas quando o controle for devolvido ao processo chamador. Tal comportamento leva à inversão de prioridade, quando, por exemplo, um processo de maior prioridade que aquele que acionou a chamada de sistema encontra-se executável, requerendo o controle da CPU.

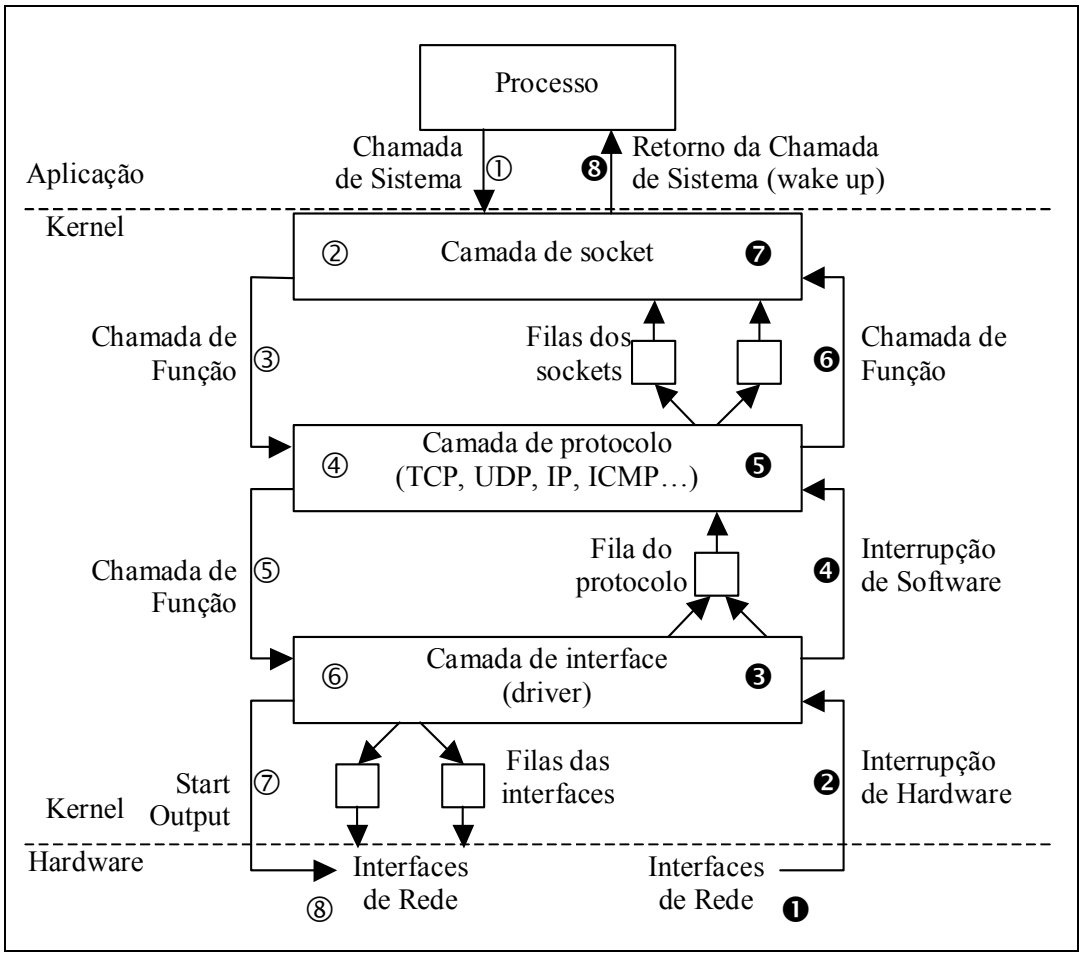


Figura 2.1 - Visão geral do processamento de rede no padrão BSD4.4

2. Chamadas de sistema (2): As chamadas de sistema dos GPOS bloqueiam a execução do processo e podem causar seu reescalonamento, se o sistema de comunicação não está pronto para o envio ou se os dados ainda não chegaram para a recepção. Observa-se, então, que existem

várias mudanças de contexto que poderiam ser evitadas se o processo envolvido na comunicação invocasse a chamada de forma assíncrona ou somente no momento em que a operação pudesse ser completada. Mudanças de contexto são caracterizadas por operações de armazenamento e recuperação dos registradores e tabelas de paginação do processo interrompido e do processo acordado, respectivamente. Tais operações podem representar perda de eficiência quando executadas com maior freqüência, principalmente por envolverem acesso à memória virtual, em alguns casos9. O suporte a chamadas de sistema assíncronas levaria a uma boa redução no número de mudanças de contexto.

| **Envio de dados** | **Recepção de dados** |
|---|---|
| ① O processo remetente solicita o envio dos dados através de uma **chamada de sistema**, cujos parâmetros são um apontador para o buffer de dados e o descritor do socket. | ❶ A interface recebe os dados armazenando em seu buffer de recepção, até completar o quadro Ethernet. |
| ② O *kernel* copia os dados do buffer para um mbuf chain alocado de um pool. | ❷ Ao completar o quadro, a interface dispara uma **interrupção de hardware**, que é tratada pelo *kernel*, escalonando o driver da interface. |
| ③ Por uma chamada de função, é acionada a rotina de envio do protocolo correspondente ao tipo do socket informado. | ❸ O driver do dispositivo copia os dados para um mbuf chain, alocado a partir do pool, retirando o cabeçalho da camada de enlace. |
| ④ O processamento da camada de protocolo envolve a adição de cabeçalhos aos mbufs, além de operações como checksum, que levam à leitura dos dados dos mbufs algumas vezes. | ❹ O driver identifica o protocolo correspondente ao quadro, coloca os mbufs na fila do protocolo e dispara uma **interrupção de software**, tratada pelo *kernel* escalonando a recepção do protocolo para execução. |
| ⑤ A camada de protocolo, após determinar a interface de saída, faz a chamada à rotina de envio do driver do dispositivo, passando o ponteiro para os mbufs. | ❺ A recepção na camada de protocolo envolve a retirada de cabeçalhos dos mbufs e outras operações que exigem leitura dos dados. |
| ⑥ O driver de dispositivo complementa o quadro de enlace e coloca os mbufs na fila de envio da interface. | ❻ O mbuf chain é colocado na fila do socket correspondente à porta local especificada. |
| ⑦ Se a interface não estiver ocupada, o driver faz a chamada da função “start output” da interface diretamente, senão o mbuf ficará na fila até que os primeiros sejam enviados. | ❼ Quando o processo toma posse da CPU, a camada socket cria um mbuf identificando a origem dos dados. |
| ⑧ A interface, depois de processados os quadros predecessores na fila, copia os mbufs para seu buffer de transmissão e inicia o envio. | ❽ O processo receptor é acordado, pois estava bloqueado pela **chamada de sistema** de recepção. Quando escalonado para execução, os dados dos mbufs são copiados para o buffer do processo. |

Tabela 2.1 - Descrição do processamento de rede no padrão BSD4.4

[9] Se as páginas de memória necessárias para o novo contexto de execução não mais se encontram na memória principal, devem ser recuperadas a partir da memória virtual.

3. Filas de pacotes: As filas de pacotes do tipo first come first served (FCFS) não garantem que os processos de alta prioridade tenham seus pacotes enviados com preferência sobre os de outros processos. De forma ideal, o escalonamento de pacotes de envio deve seguir a prioridade definida pelo escalonamento de processos ou aquela definida pela própria aplicação.

4. Utilização de interrupções de hardware: A manipulação de interrupções de hardware é uma das tarefas de mais alta prioridade em um sistema operacional, capaz de interromper qualquer processo de usuário, mesmo que ele não seja o responsável pela ação do dispositivo. Além disso, o tempo de CPU consumido no tratamento da interrupção é atribuído ao processo que foi interrompido, causando inversão de prioridade e anomalias no escalonamento, já que o tempo de CPU é contado como fator nos algoritmos de escalonamento.

5. Utilização de interrupções de software: A manipulação de interrupções de software está abaixo das interrupções de hardware na escala de prioridades de execução do sistema, mas acima da execução de processos de usuário. Isso leva aos mesmos problemas citados no item anterior, como a contabilidade inapropriada do tempo de uso dos recursos.

6. Descarte tardio de pacotes: O descarte de pacotes, como meio de diminuir a sobrecarga do receptor, pode ocorrer somente depois que muitos recursos de CPU foram investidos no tratamento do pacote descartado, levando a uma perda de capacidade de processamento da CPU.



## 2.3 Modelagem do sistema

A partir de um estudo sobre o funcionamento de sistemas operacionais, pode-se concluir que existem dois fluxos principais que comandam aplicações distribuídas:

- Fluxo de dados: é a seqüência composta por unidades de informação que são transmitidas entre threads. Essa transmissão pode envolver estações distintas, comunicando-se através de enlaces de rede, ou ocorrer internamente em uma mesma máquina, por meio de buffers ou parâmetros em chamadas de função (e.g. uma comunicação entre níveis de protocolo adjacentes).
- Fluxo de instruções: é a seqüência composta por comandos a serem interpretados por microprocessadores, que definem uma aplicação em execução ou uma rotina de controle do sistema, como uma entidade de protocolo.

Em sistemas operacionais multimídia, é necessário garantir ao fluxo de dados, alvo principal da provisão de QoS, que suas necessidades de utilização dos recursos de comunicação sejam respeitadas ao longo do caminho entre aplicação[10] e rede. Em várias etapas desse caminho, porém, é observada uma dependência entre o fluxo de dados e a capacidade de processamento de toda a pilha de protocolos, além da aplicação. Assim, o processamento do fluxo de instruções pela CPU deve ter garantias análogas àquelas dadas ao fluxo de dados, de forma que a interferência de um fluxo sobre o outro seja contabilizada para a gerência e controle da QoS como um todo.

Para visualizar esse relacionamento em um sistema operacional de uma forma genérica, considerando as arquiteturas aqui já descritas e as que serão apresentadas, é de grande utilidade a definição de um modelo de funcionamento do sistema. As técnicas de modelagem baseadas em redes de filas estendidas (Soares, 1990) podem ser utilizadas para descrever o comportamento dos fluxos de dados e de instruções em um sistema operacional, conforme ilustrado na Figura 2.2.

Algumas premissas foram adotadas para compor a generalidade do modelo. A principal delas é a definição de que cada nível da pilha de protocolos é executado por um processo em separado, dedicado a um certo fluxo de dados ou

[10] Neste ponto, o termo aplicação se refere a qualquer tarefa que envolve comunicação em rede, de aplicações do usuário a tarefas de controle do sistema, como o encaminhamento de pacotes.

compartilhado entre todos os fluxos. Não há perda de generalidade nessa premissa, já que se pode abstrair que uma entidade de protocolo reúna a execução de vários níveis reais da pilha, conforme for o espaço de endereçamento[11] em que trabalham esses níveis. Se não existe a mudança de espaço de endereçamento, não existe a necessidade de filas de comunicação entre esses níveis, pois os dados são passados entre as rotinas de tratamento por meio de chamadas de função.

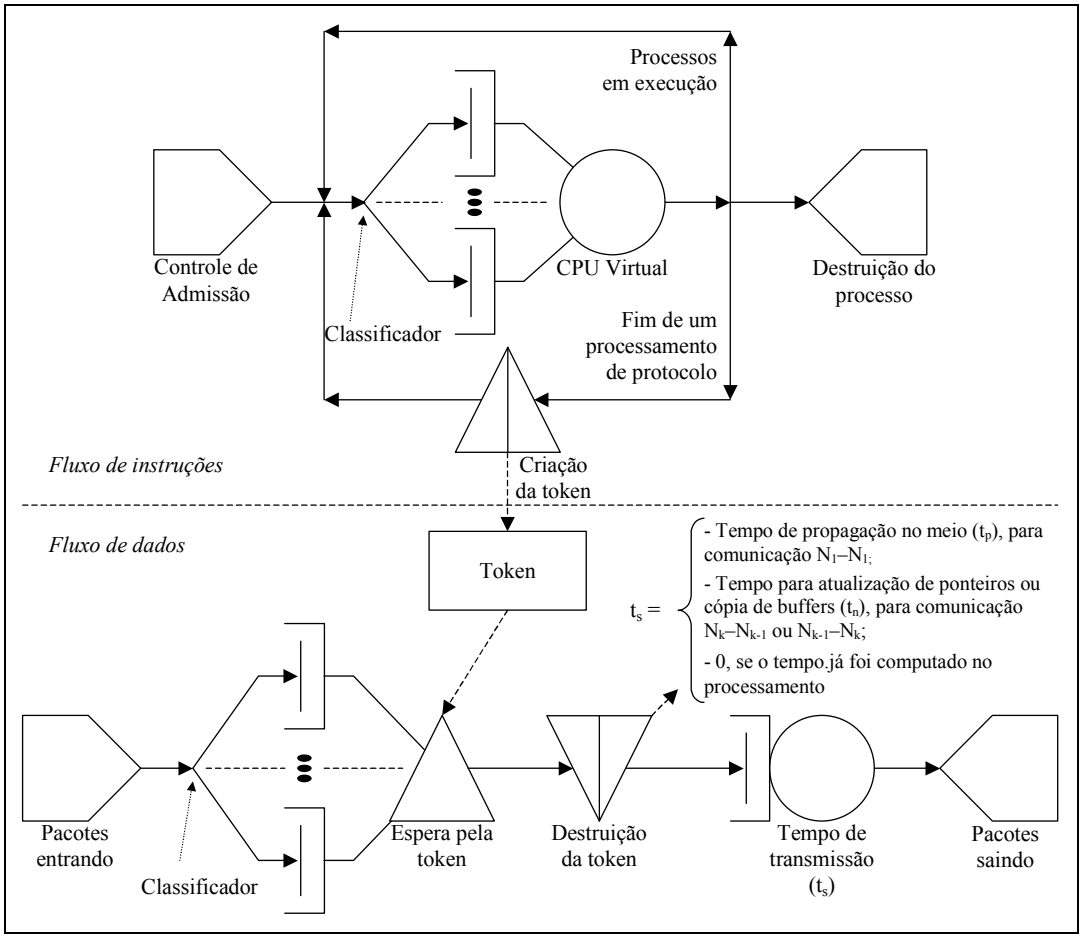


Figura 2.2 - Modelo baseado em redes de filas para sistemas operacionais em estações finais

Comparando-se alguns exemplos de arquiteturas, pode-se clarificar a descrição acima. Num GPOS, o *kernel* processa todas as camadas da pilha de protocolos em seu espaço de endereçamento, o que permite afirmar que tais camadas componham uma única entidade de protocolo, correspondendo ao contexto de um processo (não preemptivo, por ser, na realidade, o *kernel*).

[11] O espaço de endereçamento de um processo define o contexto e a área de memória exclusiva deste e de suas threads, onde podem ser feitas transferências de dados entre rotinas sem a necessidade de cópias.

Existem, portanto, filas de espera pelo processamento somente entre o hardware e o *kernel* e entre o *kernel* e a aplicação. Em sistemas em que a totalidade da pilha de protocolos está definida no espaço do usuário (Gopalakrishna, 1994), existirá um único conjunto de filas de pacotes em espera pelo processamento, entre o hardware e o processo que implementa a pilha de protocolos. Nesses sistemas, normalmente é o driver do dispositivo ou a própria interface que demultiplexa os pacotes de entrada entre as filas por socket. Não existem filas entre a aplicação e o processo do protocolo, já que eles podem estar no mesmo contexto ou, ainda, podem usar esquemas de memória compartilhada para o acesso aos dados.

Um processo em execução espera em uma das filas da CPU pelo seu escalonamento, que, quando efetuado, dá o direito de uso da CPU por um pequeno espaço de tempo. Terminado esse período, se o processo é uma entidade de protocolo e teve seu processamento completado, ele habilita o serviço de comunicação para a transferência dos dados entre os níveis correspondentes. Senão, ele volta para a fila de espera da CPU ou encerra sua execução.

O fluxo de dados, recebido do nível anterior ($N_{k-1}$ ou $N_{k+1}$) ou adjacente (no caso $N_1$-$N_1$), aguarda a execução da entidade de protocolo correspondente, mantendo-se em uma fila de espera. Ao ser liberado, é transmitido para o próximo nível, totalizando um tempo de transmissão de acordo com a situação: se a comunicação é do tipo $N_k$-$N_{k-1}$ ou $N_{k-1}$-$N_k$, o tempo será aquele necessário para a atualização de alguns ponteiros e/ou movimentação de buffers; se a comunicação é do tipo $N_1$-$N_1$, o tempo será igual ao tempo de propagação no meio.

Com esse modelo, ficam evidentes os principais recursos envolvidos na comunicação entre aplicações distribuídas e a relação de dependência entre os dois fluxos. O fluxo de dados somente é liberado quando o fluxo de instruções termina o processamento da entidade de protocolo correspondente, sendo que essa execução concorre com os demais processos existentes pelo uso da CPU.

Por exemplo, uma aplicação que exige alto desempenho para a exibição de um vídeo com boa performance deve competir pela CPU com as camadas de

protocolo que estão recebendo os seus próprios dados[12]. A coexistência de outras aplicações, como áudio, acirrará esta disputa, introduzindo também mais fluxos de dados que competirão pelas filas de espera e meios de transmissão, contra o fluxo de dados da aplicação de vídeo. Se tais recursos começarem a ficar escassos, ocorrerá a degradação da qualidade de algumas aplicações, evidenciando a importância da utilização de mecanismos para a provisão de qualidade de serviço sobre esses recursos.

Nota-se que esta importância não se dá apenas nas estações finais, mas também em alguns componentes do provedor de comunicações, como comutadores e roteadores. O mesmo tipo de relação é encontrado em tais recursos, onde um fluxo de instruções comanda a operação sobre o fluxo de dados.

## 2.4 Escalonamento de processos com qualidade de serviço

Nesta Seção, serão descritos trabalhos que visam preencher as deficiências encontradas nos GPOS quanto ao escalonamento de processos com QoS. Serão observados os mecanismos propostos com o objetivo de serem considerados na modelagem da arquitetura de frameworks aqui proposta.

### 2.4.1 Hierarchical Start-time Fair Queuing

O particionamento hierárquico da largura de banda da CPU (Goyal, 1996b) é uma das formas de suportar o uso de diferentes algoritmos de escalonamento por diferentes aplicações e prover o isolamento entre suas categorias. A hierarquia é representada por uma estrutura em árvore, onde cada thread[13] pertence a exatamente um nó folha. Cada nó folha representa um agregado de threads e,

[12] Nota-se que nos GPOS, como já comentado, não existe a concorrência entre o processo que envia os dados com o seu processamento de protocolos. Porém, a recepção guiada por interrupções pode provocar a preempção de qualquer processo mesmo que não relacionado aos dados sendo recebidos.

[13] O termo thread é muito difundido nos trabalhos sobre sistemas operacionais, mesmo em línguas diferentes do inglês, dada a dificuldade de tradução direta. Ao longo deste trabalho o termo thread será utilizado para representar qualquer entidade de processamento escalonável.

portanto, uma categoria de aplicações. Cada nó interno na árvore representa um agregado de categorias de aplicações. Todos os nós possuem um peso que determina a percentagem de largura de banda que deve ser alocada para a classe de aplicações representada por esse nó, a partir de seu nó pai. Além disso, cada nó possui um escalonador: o escalonador de um nó folha escalona todas as threads pertencentes ao nó folha; o escalonador de um nó intermediário escalona seus nós filhos.

A Figura 2.3 ilustra uma possível estrutura de escalonamento. Nela, a classe root tem três subclasses: tempo real severo, temo real suave e melhor esforço, com os pesos 1, 3 e 6, respectivamente. A largura de banda da classe melhor esforço foi dividida igualmente entre as classes folhas usuário1 e usuário2, com pesos iguais a 1. Esta distribuição de pesos equivale à descrição de que 10% da largura de banda da CPU será destinada à categoria de aplicações de tempo real severo, 30% à categoria de tempo real suave e 60% à classe de aplicações de melhor esforço; desses 60%, 50% estão reservados para as threads de usuário1 e 50% para as threads pertencentes a usuário2. Enquanto as classes de tempo real suave e usuário1 executam um algoritmo de escalonamento de distribuição justa, as classes tempo real severo e usuário2 utilizam os escalonadores EDF e de tempo compartilhado, respectivamente.

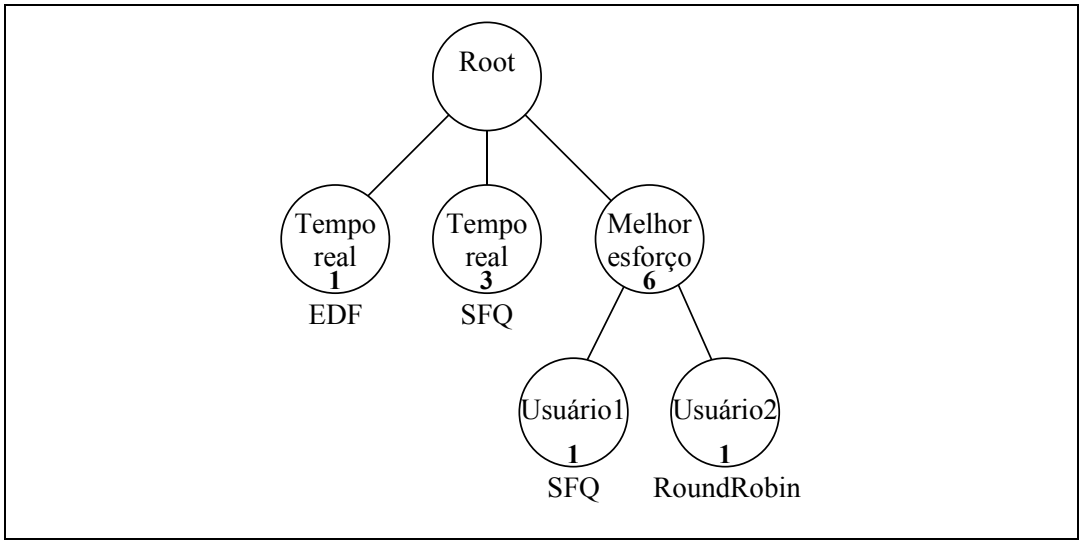


Figura 2.3 - Exemplo de estrutura de escalonamento

Observa-se, então, que os escalonadores dos nós folhas da hierarquia são determinados de acordo com as necessidades das aplicações. Para o

escalonamento dos nós intermediários foi proposto o algoritmo SFQ, responsável por garantir a total justiça do uso da CPU entre as classes de aplicações. Em sistemas com alto grau de adaptabilidade, porém, pode ser desejável a definição de um novo escalonador também para nós intermediários em alguns ramos da árvore. O uso do SFQ se tornaria apenas um caso particular em uma arquitetura realmente adaptável como a proposta no presente trabalho.

(Goyal, 1996b) propõe, ainda, que a infra-estrutura de escalonamento seja usada por um *gerenciador de QoS* (Figura 2.4), para o qual as aplicações irão direcionar suas *solicitações*, especificando suas necessidades. O gerenciador de QoS deve ser capaz de determinar os recursos necessários para atingir as requisições de QoS das aplicações e decidir a qual *classe de escalonamento* a aplicação deve pertencer. O mesmo gerenciador poderá executar procedimentos de *controle de admissão*, para determinar se as requisições por recursos podem ser satisfeitas e *alocar os recursos* para a aplicação na classe apropriada.

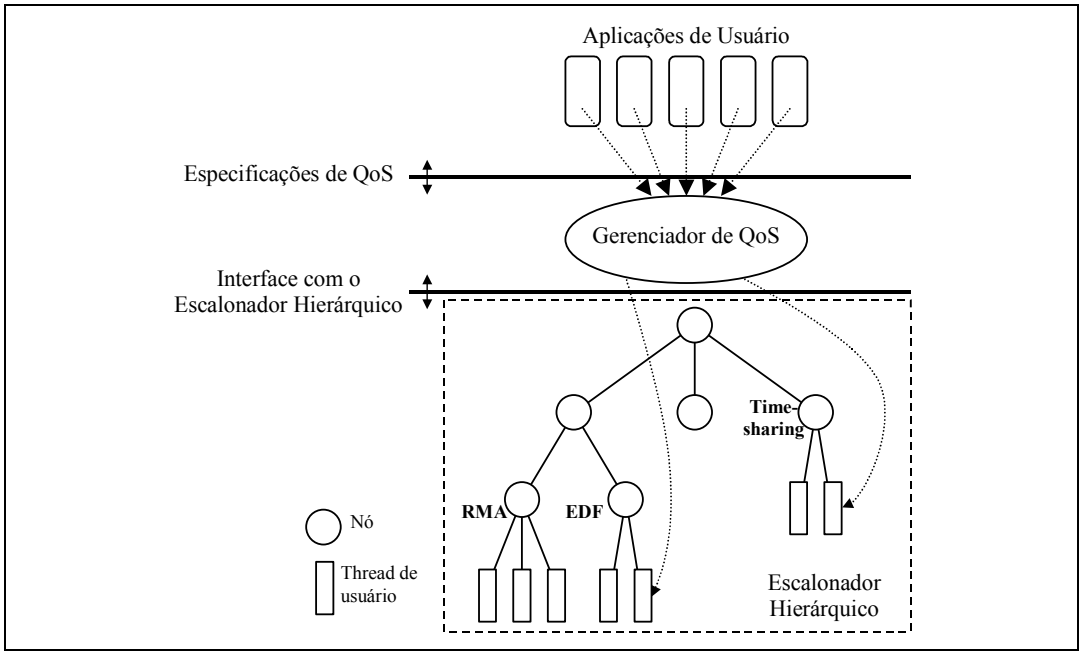


Figura 2.4 - Interação do Gerenciador de QoS com o escalonador hierárquico

Além dessas tarefas, o gerenciador pode mover as aplicações entre as classes ou modificar a alocação de recursos em resposta a uma mudança nas necessidades de QoS ou na situação de carga (*renegociação e sintonização de QoS*). O trabalho citado não abrangeu a programação do gerenciador de QoS, deixando em aberto as formas de implementação dessas políticas e o conjunto de

parâmetros de QoS das requisições. A arquitetura proposta pelo presente trabalho inclui uma discussão detalhada sobre esse conjunto de mecanismos, apresentada no Capítulo 3.

### 2.4.2 Outros escalonadores hierárquicos

Em (Ford, 1996), é proposto um escalonamento de CPU por herança (CPU Inheritance Scheduling), um framework no qual threads arbitrárias podem agir como escalonadores para outras threads. Uma thread pode, então, "doar temporariamente" seu tempo de CPU para as threads selecionadas, enquanto esperam por eventos de seu interesse, como interrupções de clock ou timer. A thread que recebe este tempo de CPU pode passá-lo para outras threads, formando uma hierarquia lógica de escalonadores.

(Regehr, 2001) propõe uma infra-estrutura de escalonadores hierárquicos carregáveis em tempo de execução (Hierarchical Loadable Scheduler), provendo uma interface de programação bem definida para a construção de escalonadores. Esses, por sua vez, são notificados sobre todos os eventos do sistema operacional que podem demandar decisões de escalonamento, devendo estar aptos a tomar a ação apropriada. Para isso, foi definido um modelo de programação de escalonadores carregáveis, formalizando as suas possíveis interações com o *kernel*. O modelo inclui quando e por que diferentes notificações são enviadas aos escalonadores, quais as ações que eles podem tomar ao respondê-las, como é feito o controle de concorrência na infra-estrutura e propõe um modelo de confiança para os escalonadores carregáveis. Um protótipo foi criado sobre o *kernel* do sistema operacional Windows 2000.

A estrutura de escalonamento hierárquico será apresentada de forma genérica no presente trabalho, de forma a abranger outros recursos do sistema, além da CPU, através do conceito de *árvores de recursos virtuais*, introduzido no Capítulo 3.

### 2.4.3
### Meta-algoritmo para Políticas de Escalonamento

O meta-algoritmo LDS – Load Dependent Scheduling (Barria, 2001) foi desenvolvido com o objetivo de se tornar uma ferramenta para a análise e comparação do desempenho de diversos algoritmos de escalonamento. Ele é capaz de emular a operação de vários algoritmos, a partir da modificação dos seus próprios parâmetros de operação. A implementação do LDS para o escalonamento de recursos seria vantajosa para a provisão de QoS, já que vários algoritmos que oferecem certas garantias de processamento podem ser por ele emulados (e.g. weighted fair queuing (Demers, 1990) – WFQ).

Além disso, o LDS introduz mais um ponto de *adaptabilidade* ao sistema, onde a adição de novas estratégias de escalonamento pode ser feita simplesmente através da mudança dos valores que controlam a operação do algoritmo. Nota-se que esse esquema de adaptação possui uma vantagem sobre a inserção de módulos no *kernel*, por não exigir a execução de etapas anteriores para a implementação das estratégias, como programação e geração de código objeto.

Por outro lado, o LDS não possibilita a utilização simultânea de mais de uma política de escalonamento, o que impede o atendimento satisfatório a múltiplas categorias de aplicações. Além disso, o algoritmo não é capaz de emular políticas que envolvem a determinação de prazos limites para execução, como os algoritmos de escalonamento para aplicações de tempo real severo. Um último problema pode estar no grande número de entradas da tabela para caracterizar todas as possíveis situações de carga de recursos como a CPU.

A utilização do algoritmo LDS em uma estrutura hierárquica de escalonamento é uma proposta que possibilitaria a criação de diferentes estratégias para as categorias de aplicações. Da mesma forma, o número de entradas na tabela seria reduzido, já que existiria uma tabela para cada categoria, caracterizando uma situação local de carga. A deficiência para emulação de algoritmos de tempo real pode ser suprida pelo uso do LDS nos escalonadores internos da hierarquia, deixando para os escalonadores folhas a implementação de algoritmos mais específicos, quando houver essa necessidade.

O LDS opera ciclicamente (Figura 2.5), onde cada ciclo é composto por *N+1* fases (*N* corresponde ao número de filas de execução da CPU). A fase $f_0$ corresponde a uma fase de inatividade. Durante a fase $f_i$ ($1 \leq i \leq N$), a CPU atenderá exclusivamente as threads da fila *i*, sendo que a duração da fase depende do estado que caracteriza a carga da CPU. Assim, o tamanho do quantum ou, se preferível, o número de quanta que serão atribuídos à execução de cada fase (threads de uma mesma categoria) pode variar em diferentes ciclos de operação do LDS.

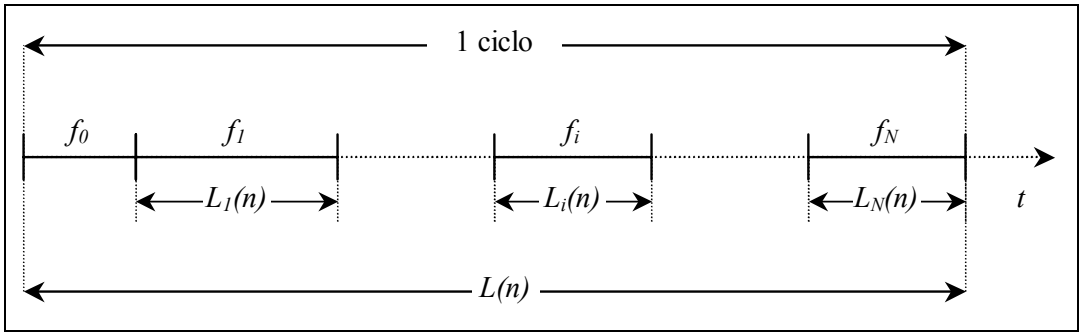


Figura 2.5 - Diagrama temporal de um ciclo de execução do algoritmo LDS

Os dados que compõem a situação de carga da CPU (número de threads em execução em cada fila da CPU) indexam uma tabela para a localização do vetor que descreverá a duração de execução de cada thread no ciclo sendo iniciado. A tabela LDS (Tabela 2.2), como é chamada, possui *2N* colunas, onde *N* corresponde à quantidade de filas atendidas pela CPU. As primeiras *N* colunas são rotuladas por $n_i$, indicando o número de threads em execução em cada fila da CPU no início do ciclo. As *N* colunas restantes são rotuladas com $L_i(n)$, indicando o número máximo de quanta que pode ser atribuído para a execução da fase $f_i$. A quantidade de linhas da tabela é igual ao número de combinações lógicas diferentes que podem caracterizar a carga da CPU.

| $N_1$ | $N_2$ | $N_3$ | | $N_N$ | $L_1(n)$ | $L_2(n)$ | $L_3(n)$ | | $L_N(n)$ |
|---|---|---|---|---|---|---|---|---|---|

Tabela 2.2 - Cabeçalho da Tabela LDS

O algoritmo LDS pode ser descrito em alto nível como a seguinte seqüência de passos:

1. No início de um ciclo, é determinado o vetor de estado n, caracterizando a carga da CPU.

2. Se n=0, deve-se colocar a CPU em estado ocioso, esperando o início da execução de alguma thread (fase f0). Quando for iniciada a execução de alguma thread, inicia-se a fase de atendimento àquela que chegar primeiro.

3. Se n≠0, são lidas as N primeiras colunas de cada linha da tabela LDS, até que se encontre a situação de carga atual da CPU. Encontrada a linha correspondente, são lidas as próximas N colunas, especificando o valor de Li(n).

4. A seguir, é dado o controle da CPU às threads da fase f1, por tantos quanta quantos os especificados por L1(n). Em seguida, são atendidas as threads da fase f2, e assim por diante, até a fase fN.

5. Durante a fase fi, as threads da fila i são atendidas pela CPU de forma exclusiva, pelo período determinado pelo número de quanta Li(n). Se durante essa fase as threads entram em estado bloqueado ou encerram, a fase fi deve ser abortada para o início da fase fi+1.

## 2.5 Subsistema de rede com qualidade de serviço

Para buscar a solução dos problemas encontrados no subsistema de rede dos GPOS, muitos trabalhos definem a criação de novos sistemas operacionais baseados, principalmente, na arquitetura de *microkernel*.

Nessa alternativa aos sistemas monolíticos, o *kernel* do sistema é muito pequeno, responsável por comandar a passagem de mensagens entre os módulos do sistema e o hardware, além de funções básicas como trocas de contexto entre processos. Algumas funções, como escalonamento de processos e controle de entrada e saída, podem estar implementadas no núcleo, mas a idéia é mantê-lo com o menor tamanho possível. A tradução para uma outra arquitetura de hardware envolveria apenas a modificação código do *microkernel*, já que os módulos não têm dependência direta do hardware, facilitando a portabilidade do sistema. A adaptabilidade a novos serviços é beneficiada, já que um módulo do sistema é, geralmente, implementado por um processo que se comunica com o

*microkernel*, bastando para a sua substituição o encerramento do processo antigo e o disparo do novo. Outra vantagem observada está no carregamento apenas dos módulos necessários no momento, sem a ativação daqueles que não serão utilizados, representando uma boa economia de memória e processamento.

Uma das críticas ao *microkernel* é a baixa velocidade na comunicação entre os módulos, por ser baseada em troca de mensagens. A arquitetura de passagem de mensagens tem sua eficiência comprometida principalmente pela sobrecarga de mensagens geradas. Questões de segurança na substituição e inserção de módulos também devem ser observadas, já que um módulo mal escrito ou programado por um usuário mal intencionado pode afetar a estabilidade do sistema. Finalmente, os sistemas baseados em *microkernel* não exploram funções específicas do hardware por definir seus drivers em módulos que não se relacionam diretamente a ele. Essas funções podem incluir alternativas de acesso direto ao dispositivo, ou instruções especiais que representam um incremento no desempenho desse acesso.

Nesta Seção, serão apresentados alguns sistemas baseados em *microkernel* para a provisão de QoS com ênfase no subsistema de rede, sendo que alguns deles definem, também, soluções para o escalonamento de processos. Outros trabalhos que serão descritos visam a solução de problemas isolados no processamento da pilha de protocolos de rede dos GPOS.



### 2.5.1 SUMO

O projeto SUMO (Support for Multimedia in Operating Systems) (Coulson, 1995) da Universidade de Lancaster engloba a implementação de um sistema operacional baseado em um *microkernel* – Chorus (Campbell, 1996) – oferecendo facilidades para o suporte a aplicações multimídia distribuídas. A arquitetura do sistema é baseada em três novos princípios:

- As aplicações são guiadas por “upcalls”, definindo que os eventos de comunicação são iniciados pelo sistema e não pela aplicação;

- As funções de gerenciamento de recursos são executadas cooperativamente entre o *kernel* e componentes do nível de usuário (sistema split-level);
- A transferência de controle é feita separadamente da transferência de dados, de forma assíncrona.

Além disso, estruturas e mecanismos para o gerenciamento de recursos com QoS são definidos. A seguir, serão abordadas essas características.

*Aplicações guiadas por upcalls*

A infra-estrutura do sistema SUMO foi projetada para que trabalhasse de forma ativa, deixando as aplicações com um processamento passivo. Isso significa que, no estabelecimento de uma conexão, o sistema, e não a aplicação, dispara as threads responsáveis por tratar a comunicação e aloca os buffers para as conexões. No momento da transferência de dados, é o sistema quem decide por ativar a aplicação através de uma chamada (upcall), nos instantes determinados pela *especificação de QoS* fornecida pela aplicação.

Neste tipo de interação entre *kernel* e aplicação, como o sistema realiza o escalonamento do processo de acordo com a comunicação, a *monitoração* e o *gerenciamento de QoS* para a conexão são providos com maior facilidade. Por exemplo, não existe a necessidade de policiamento e moldagem do fluxo gerado pela aplicação, pois esta não consegue enviar dados fora do tempo concedido pela sua especificação de QoS. Além disso, é reduzido o número de mudanças de contexto, pois o processo é ativado quando existem os dados a serem recebidos ou quando os dados por ele gerados podem ser enviados. Outra vantagem da arquitetura está no modelo de programação orientado a eventos em que é baseada, por ser ideal para a estruturação de aplicações distribuídas de tempo real. O programador declara como os eventos de comunicação serão tratados, mas o momento de execução deste tratamento é governado pelos parâmetros de QoS fornecidos em tempo de conexão.

*Sistema split-level*

A estruturação split-level define que as funções-chave do sistema devem ter seu desempenho compartilhado cooperativamente entre o kernel e o nível de usuário, intercomunicando-se de forma assíncrona para a troca de informações de gerenciamento. Assim, tira-se proveito da minimização de overheads de comunicação conseguida através da implementação de várias funções do sistema no mesmo espaço de endereçamento da aplicação.

No escalonamento de processos split-level (Govindan, 1991), um pequeno número de processadores virtuais executa threads de usuário em cada espaço de endereçamento, seguindo as definições: i) cada escalonador de nível de usuário (user-level scheduler – ULS) sempre escolhe a thread de usuário mais urgente; ii) cada escalonador de nível de *kernel* (kernel-level scheduler – KLS) sempre escolhe o processador virtual que possui a thread de usuário mais urgente no contexto geral. A arquitetura permite, dessa forma, mudanças de contexto baratas entre threads no mesmo espaço de endereçamento, ao mesmo tempo em que as urgências relativas entre threads pertencentes a outros espaços de endereçamento são asseguradas.

Na comunicação split-level, o *kernel* fica responsável por multiplexar e demultiplexar os pacotes de rede entre os espaços de endereçamento das aplicações, deixando que cada uma delas realize o processamento do nível de transporte. O processamento da comunicação pode, então, tirar vantagem do escalonamento split-level, pela implementação das threads de comunicação no mesmo espaço de endereçamento da aplicação. Dessa maneira, as mudanças de contexto entre threads de processamento e de comunicação terão baixo custo, não haverá a necessidade de cópias de dados e o processamento da comunicação poderá seguir a urgência da aplicação correspondente.

*Separação da transferência de controle da transferência de dados*

Nos GPOS, as transferências de controle e de dados estão acopladas, como no caso de uma chamada de sistema em que os dados são passados para o *kernel* ao mesmo tempo em que o controle de execução é transferido. No SUMO, existe

a separação da transferência de controle da transferência de dados, facilitando a implementação de chamadas de sistema e interrupções de software assíncronas, por exemplo. A definição de chamadas de sistema assíncronas é interessante por reduzir, também, o número de mudanças de contexto provocadas por processos que antes seriam bloqueados na espera por uma resposta às suas requisições.

*Gerenciamento de recursos*

(Campbell, 1996) define dois estágios que antecedem uma efetiva reserva de recursos em sistemas operacionais. O *mapeamento de QoS* é o processo de transformação dos parâmetros descritos no contrato de serviço (*especificação de QoS*) em valores que representam as reais necessidades de desempenho para cada um dos recursos envolvidos. Por exemplo, o subsistema de rede pode oferecer garantias sobre a largura de banda, retardo máximo e taxa de perda de pacotes. Esses valores são calculados a partir dos parâmetros informados na especificação de QoS. Para uma aplicação de vídeo, a largura de banda pode ser calculada a partir da taxa de quadros de vídeo e do tamanho do quadro. O retardo máximo é obtido através da parcela atribuída à estação pelo protocolo de negociação, proveniente da distribuição da latência total informada na especificação. A taxa de perda de pacotes é calculada a partir de parâmetros de mais alto nível como perda de quadros e intervalo entre perdas.

A *verificação de admissão* determina se recursos suficientes para atender aos parâmetros descritos acima estão disponíveis no sistema. Por exemplo, o subsistema de rede de cada nó que define o caminho do fluxo a ser admitido deve realizar três diferentes verificações: teste de largura de banda; teste de retardo máximo e teste de disponibilidade de buffers.

O gerenciador de fluxos ilustrado na Figura 2.6 é uma entidade responsável por administrar as necessidades locais no sistema operacional para os fluxos de rede, distribuindo a responsabilidade da gerência de QoS entre os módulos de gerenciamento individuais de recursos (CPU, memória e rede). O gerenciador de fluxos utiliza um protocolo de gerenciamento de fluxos para obter a parcela da QoS especificada que deve por ele ser negociada localmente.

Dessa forma, quando uma conexão com QoS é solicitada, a função de mapeamento traduz os parâmetros de nível do usuário em parâmetros a serem considerados por cada um dos gerenciadores de recursos, inclusive o gerenciador de fluxos. Com base na distribuição da alocação de recursos entre os nós participantes da comunicação, o gerenciador de fluxos interage com os gerenciadores de CPU, memória e rede para que sejam feitas as verificações de admissão, de forma independente. Se todos eles responderem positivamente sobre a viabilidade da reserva requerida, a conexão pode ser aceita, caso contrário pode ser feita uma renegociação. O gerenciador de fluxos é, ainda, responsável pelo ajuste dinâmico das reservas de recursos (*sintonização de QoS*), em resposta a ocorrências de degradação da QoS no momento da provisão do serviço.

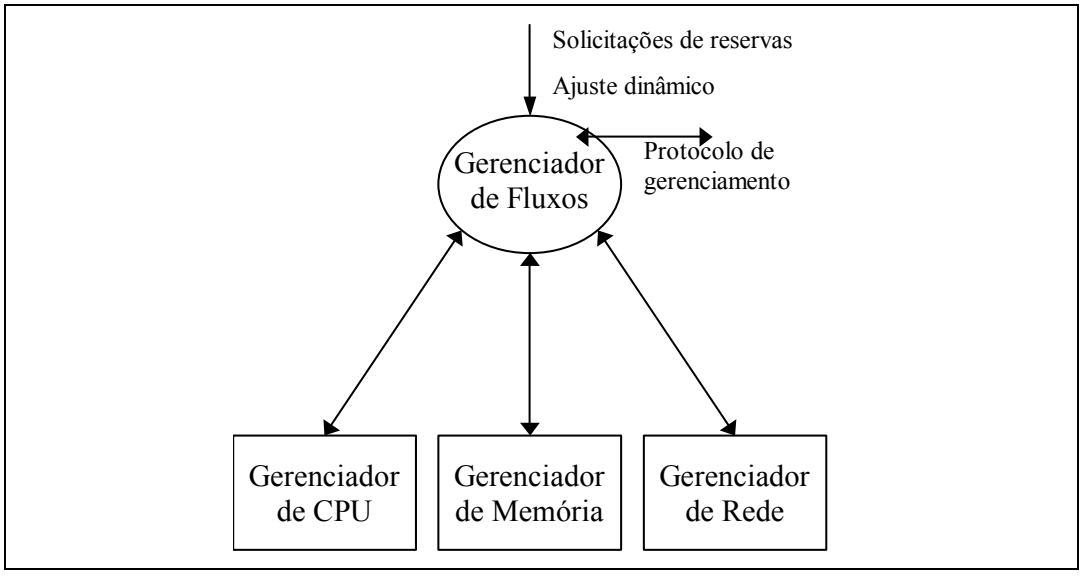


Figura 2.6 - Gerenciador de fluxos proposto por (Campbell, 1996)



### 2.5.2 Nemesis

O projeto Nemesis (Leslie, 1996) da Cambridge University é um exemplo de sistema operacional estruturado verticalmente (Black, 1997), onde as aplicações realizam a maior parte possível do processamento a elas pertinente, ao invés de passar para o *kernel* ou para servidores esse trabalho. As APIs que implementam as funções do sistema operacional são disponibilizadas como bibliotecas compartilhadas. Existe apenas um espaço de endereçamento virtual, sem perda da proteção de memória entre os domínios de aplicação (processos).

Esse espaço único reduz o número de mudanças de contexto, claramente vantajoso para aplicações com fortes requisitos de QoS, como já visto.

O *microkernel* Nemesis consiste apenas do escalonador e de manipuladores de interrupções e traps, não havendo threads de *kernel*. A Figura 2.7 mostra a organização do Nemesis como um conjunto de domínios e um pequeno *kernel*. O projeto considera que as interfaces de rede são capazes de demultiplexar os dados recebidos, colocando-os diretamente na fila de destino correspondente. A aplicação contém um protocolo estruturado verticalmente para cada fluxo de comunicação, estabelecendo um mesmo espaço de endereçamento para aplicação e processamento de protocolos.

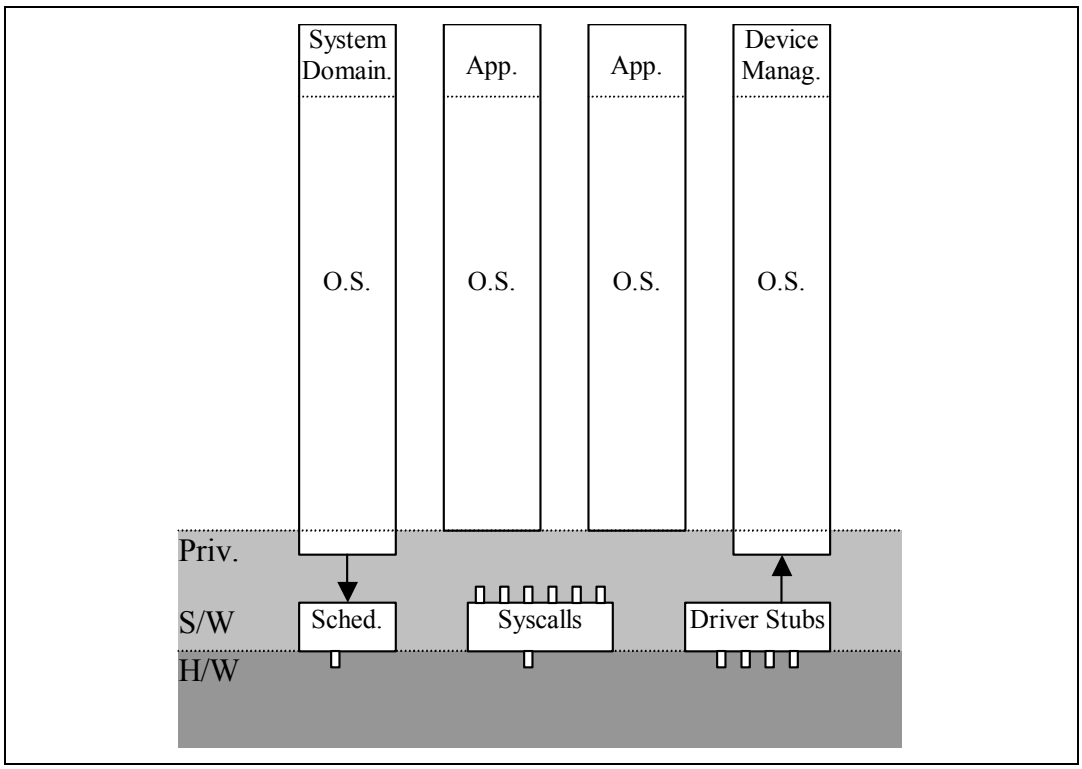


Figura 2.7 - Arquitetura do sistema Nemesis

O sistema Nemesis possui uma arquitetura de gerenciamento de recursos (Oparah, 1998) cujo elemento central é denominado Gerenciador de QoS, representado por um serviço do sistema e responsável por coordenar a distribuição dos recursos entre os domínios de aplicação. A fim de obter garantias de QoS ou renegociar uma garantia antiga, as aplicações devem encaminhar uma requisição ao gerenciador de QoS que, baseado na disponibilidade dos recursos, distribui suas decisões de alocação entre os gerenciadores de recursos. Cada recurso do

sistema operacional, como CPU, buffers de comunicação e disco, possui um gerenciador de recurso designado para assegurar que as garantias solicitadas sejam respeitadas pelo escalonamento apropriado do recurso.

O gerenciador de QoS provê, também, uma interface gráfica para o usuário, para que o estado atual de alocação de recursos para cada uma das aplicações possa ser informado e ajustado. A arquitetura foi implementada através de um protótipo que oferece somente o gerenciamento da CPU. Nesse caso, o usuário pode redefinir certas garantias de alocação da CPU, como a percentagem reservada para uma dada aplicação. A Figura 2.8 ilustra de forma simplificada a arquitetura de gerenciamento de recursos do Nemesis.

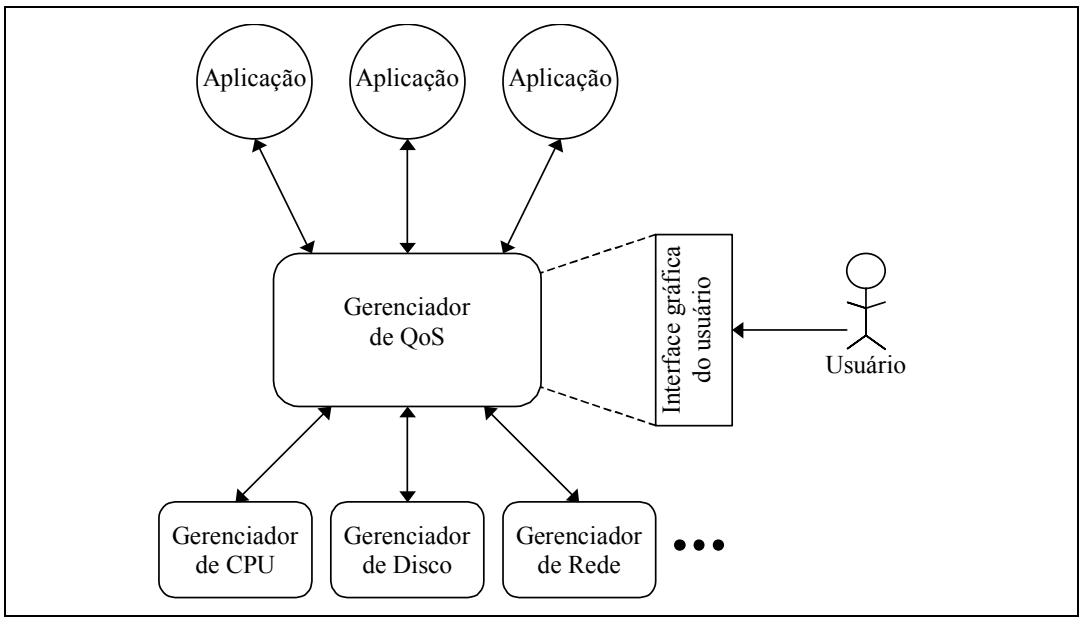


Figura 2.8 - Arquitetura de gerenciamento de recursos do Nemesis

### 2.5.3 Real Time Mach

O sistema RT-Mach (Mercer, 1994) (Lee, 1996) é uma extensão ao *kernel* Mach para o suporte a aplicações de tempo real. O antigo subsistema de rede baseado em um servidor que manipula as chamadas baseadas no padrão BSD foi substituído por uma arquitetura de processamento de protocolo no nível de usuário.

O servidor UX utilizado no Mach não preenchia as necessidades de processamento em tempo real por não suportar qualquer tipo de prioridade. Além disso, a comunicação feita entre a aplicação e esse servidor adicionava um overhead, diminuindo a vazão e aumentando a latência. Os princípios que guiaram os pesquisadores para o desenvolvimento de um subsistema de rede preditivo foram os seguintes:

1. Utilização de prioridades nos pacotes para enfileiramento;
2. Escalonamento do processamento de protocolo contra as outras atividades do sistema utilizando a prioridade dos pacotes;
3. Utilização de uma estrutura de controle de preempção para reduzir a ocorrência de inversão de prioridade e mudanças de contexto;
4. Particionamento dos recursos para eliminar a interferência entre classes de prioridade;

Assim, o grupo criou uma biblioteca que manipula o processamento de protocolo para o envio e recebimento de pacotes, interagindo com os drivers de filtragem de pacotes e de interface de rede diretamente. A biblioteca pode ser linkeditada às aplicações, de forma que elas possam fazer seu próprio processamento de protocolo em seu espaço de endereçamento. A biblioteca somente interage com o servidor UX para criar e destruir conexões e outras poucas operações.



### 2.5.4 Process per Channel

(Mehra, 1996) propõe uma arquitetura para o subsistema de comunicação baseado no modelo processo por canal, no qual cada canal é coordenado por um manipulador único e exclusivo, implementado como um processo leve (lightweight process - LWP), criado no estabelecimento do canal.

O envio de uma mensagem é feito através de uma API. Depois de feita a moldagem do tráfego, a mensagem é enfileirada na fila de mensagens do canal, para o subseqüente processamento pelo manipulador do canal. De acordo com o

tipo de canal, o manipulador é associado a uma das três filas de execução da CPU (current real time: processando mensagens que obedecem à taxa do canal; early real time: processando mensagens que violam a taxa do canal; e best-effort: canais de melhor esforço). Quando escalonado para execução (estratégia EDF), o manipulador retira cada mensagem para o processamento do protocolo, podendo ser interrompido por threads de maior prioridade. Os pacotes gerados pela segmentação da mensagem são inseridos em uma das três filas de pacotes de enlace da interface correspondente, sendo posteriormente transmitidos, de acordo com a estratégia do escalonador da interface.

Na recepção, o pacote recebido é demultiplexado diretamente para a fila de pacotes do canal correspondente, para o processamento e remontagem. O manipulador do canal é associado a uma das filas de execução da CPU e, quando escalonado, processa os pacotes da fila, podendo ser interrompido para preempção. Chegado o último pacote que completa a remontagem da mensagem, o manipulador a coloca na fila de mensagens do canal, sendo retirada quando a aplicação fizer a chamada de recepção da API.



### 2.5.5 Lazy Receiver Processing (LRP)

Na arquitetura proposta em (Druschel, 1996), chamada Lazy Receiver Processing (LRP) – Processamento tardio do receptor – alguns dos problemas de *recepção de dados* do subsistema de rede dos GPOS são resolvidos pela seguinte combinação de técnicas:

1. Substituição da fila compartilhada IP por filas por socket, de acesso direto pela interface de rede. A interface deve, então, demultiplexar os pacotes de entrada colocando-os nas filas apropriadas, de acordo com o socket de destino.
2. Execução do protocolo de recebimento executado na prioridade do processo receptor. Para isso, o processamento será executado mais tarde, no contexto da chamada de sistema disparada pelo processo responsável pelo recebimento.

Assim, o processamento do protocolo para um pacote não ocorre até que a aplicação receptora requisite-o pela chamada de sistema. Além disso, esse processamento não mais interrompe o processo em execução no momento da chegada do pacote, a não ser que o receptor tenha maior prioridade no escalonamento que o processo corrente. Evitam-se, ainda, mudanças de contexto inapropriadas, levando a um significativo aumento do desempenho do sistema.

No LRP, a própria interface de rede (seja através de firmware ou pelo driver) separa o tráfego chegando para o socket de destino e coloca os pacotes diretamente nas filas de recebimento, exclusivas de cada socket. Combinado com o processamento da pilha de protocolos na prioridade da aplicação receptora, tem-se um mecanismo de feedback para a interface sobre a capacidade das aplicações de manter o trabalho com o tráfego de entrada de um socket. Essa informação pode ser usada de forma que, quando a fila de um socket se esgota, a interface de rede possa descartar os próximos pacotes a ele destinados até que a aplicação consuma algumas posições da fila. Desse modo, a interface pode fazer o descarte sem que recursos excessivos sejam comprometidos no processamento desses pacotes a serem perdidos.

A separação do tráfego recebido pela interface de rede, combinada com o processamento na prioridade da aplicação, elimina interferências entre os pacotes destinados para sockets distintos. Adicionalmente, a latência de despacho de um pacote não é influenciada pelo subseqüente recebimento de um pacote de prioridade menor ou igual. A eliminação da fila IP compartilhada reduz fortemente a probabilidade de o pacote ser retardado ou descartado devido à ausência de recursos provocada por um tráfego destinado a um socket diferente.

A arquitetura define que o tempo de CPU gasto no processamento do protocolo de recebimento (excluindo-se o tempo para manipulação da interrupção de hardware) é contabilizado para o processo que recebeu o tráfego, fato importante já que o uso recente de CPU influencia na prioridade de escalonamento dos processos. Garante-se, então, melhor justiça no caso em que os processos de baixa prioridade recebem grandes volumes de tráfego de rede.

### 2.5.6
### Linux Network Traffic Control (LinuxTC)

As versões mais recentes de *kernel* do Linux oferecem um grande conjunto de funções de controle de tráfego de rede (Almesberger, 1999) (Radhakrishnan, 1999). As estruturas utilizadas para isso são capazes de oferecer os mecanismos necessários para o suporte às arquiteturas IntServ (Braden, 1994) e DiffServ (Blake, 1998).

De maneira geral, pode-se descrever o processamento feito pelo subsistema de rede do *kernel* do Linux, a partir da Figura 2.9. Um pacote recebido pela interface de rede é examinado, para que o *kernel* decida entre o encaminhamento para outro nó da rede e o processamento pelos protocolos de níveis mais altos da pilha. No caso de roteadores, a maioria dos pacotes será analisada para o posterior encaminhamento em uma de suas interfaces de rede. Para estações finais (hosts), os pacotes serão processados por um protocolo de transporte, tal qual UDP ou TCP. Essas camadas mais altas da pilha podem, também, gerar dados para as camadas mais baixas, para as tarefas de transmissão de dados, roteamento e encapsulamento. O encaminhamento inclui a seleção da interface de saída, a escolha do próximo salto, o encapsulamento dos pacotes, entre outros. Completada essa tarefa, os pacotes são enfileirados na respectiva interface de saída, onde entra em ação o controle de tráfego.

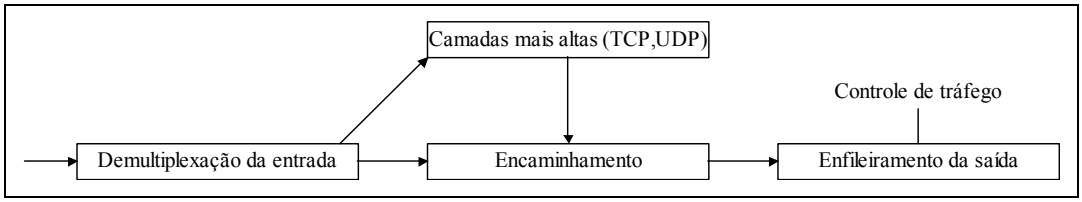


Figura 2.9 - Processamento dos dados de rede

O controle de tráfego do Linux[14] é composto pelos seguintes componentes conceituais: disciplinas de enfileiramento, classes e filtros de classificação e policiamento. Cada interface de rede tem associada sua disciplina de enfileiramento, que controla como é tratado o enfileiramento neste dispositivo.

[14] Um mecanismo semelhante está disponível para o sistema operacional Windows, através da interface Winsock2 GQoS (Bernet, 1998), porém esta API não oferece toda a flexibilidade nem a possibilidade de hierarquização do recurso como provê o LinuxTC.

Uma disciplina de enfileiramento pode ser simples tal qual aquela constituída apenas de uma única fila, como também pode ser mais elaborada e complexa, utilizando filtros para distinção dos pacotes, distribuindo-os entre classes (Figura 2.10).

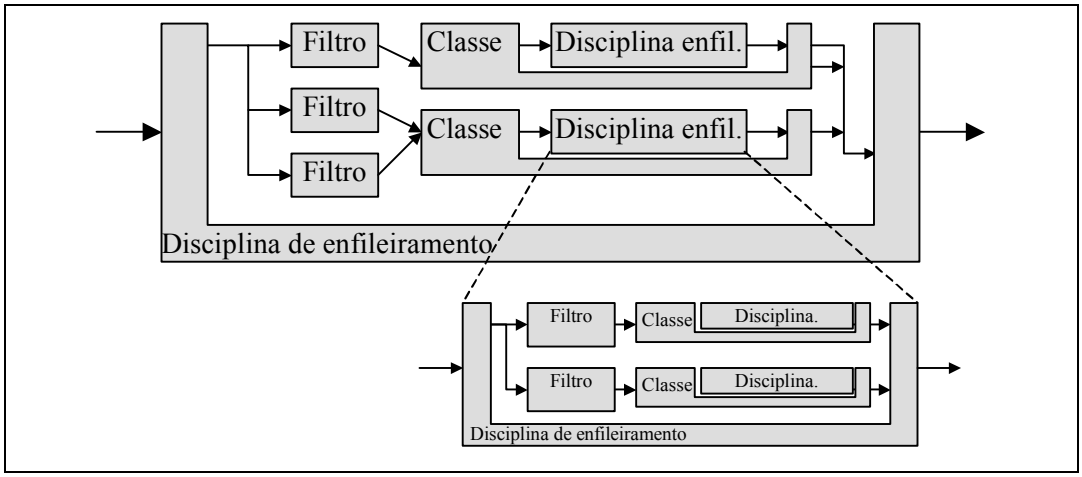


Figura 2.10 - Modelo de disciplina de enfileiramento, definido em (Almesberger, 1999)

O LinuxTC permite decidir a forma com que os pacotes devem ser enfileirados e quando eles devem ser descartados, numa situação de tráfego excedendo ao limite, por exemplo. É possível definir a ordem de envio desses pacotes, aplicando-se prioridades aos fluxos e, por último, retardar o envio de alguns pacotes para limitar a taxa de dados do tráfego de saída. Executadas todas essas operações, cada pacote pode ser entregue ao driver da interface para a transmissão pela rede.

Com este conjunto de ferramentas, é possível configurar vários tipos de políticas para o escalonamento dos pacotes, distribuídas em uma estrutura hierárquica. Cada classe de uma disciplina de enfileiramento pode possuir uma nova disciplina para a qual será delegado o escalonamento de pacotes pertencentes àquela classe. Essa estrutura hierárquica pode ser visualizada de uma forma mais clara através da Figura 2.11, onde as classes podem ser vistas como conectores entre as disciplinas de enfileiramento. O conceito de árvore de recursos virtuais, que será introduzido no Capítulo 3, é genérico o suficiente para abranger também a utilização do controle de tráfego do Linux em uma instanciação da arquitetura proposta por este trabalho.

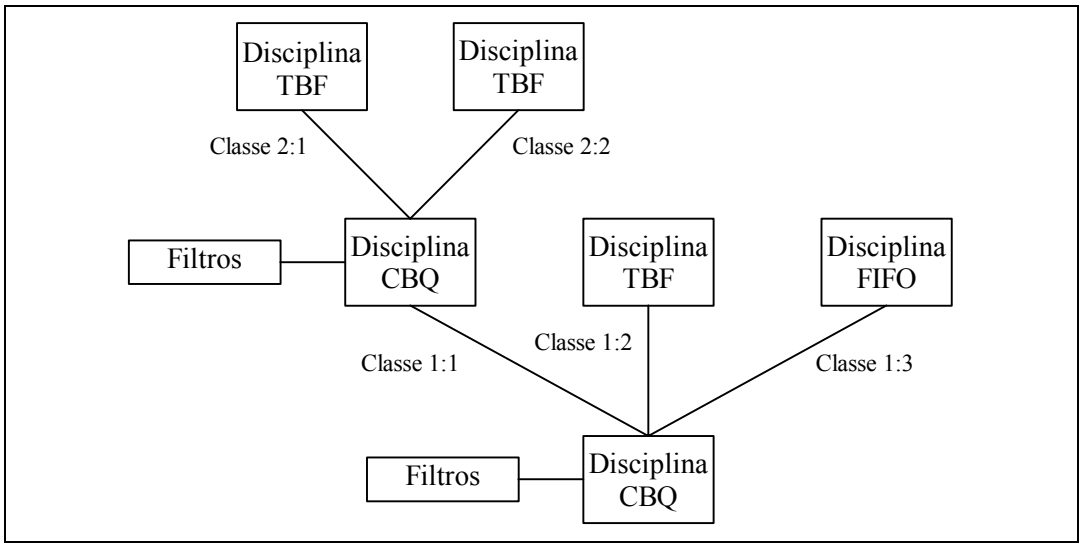


Figura 2.11 - Exemplo de hierarquia de disciplinas de enfileiramento, utilizando uma notação em árvore

## 2.6
## Suporte à Adaptabilidade no Linux

Como já definido, o conceito de adaptabilidade está relacionado à capacidade que possui um provedor para a configuração e provisão de novos serviços de comunicação e processamento, através da inclusão ou modificação de estruturas como, por exemplo, as que definem novas políticas de provisão de qualidade de serviço, em tempo de execução.

Em sistemas operacionais, são encontradas duas formas básicas para a introdução de novos serviços: através do uso de algoritmos configuráveis, que permitem uma mudança de seu próprio comportamento a partir da modificação de seus parâmetros de operação; e através da inserção ou substituição de partes de código no conjunto de instruções que compõem o gerenciamento de recursos do sistema.

O algoritmo LDS, já apresentado, é um exemplo da primeira forma de adaptabilidade. A segunda forma é encontrada freqüentemente em sistemas baseados em *microkernel*, facilitada pela arquitetura do sistema, discutida anteriormente. Já em sistemas monolíticos, é observada a dificuldade de introdução de módulos ao *kernel* em tempo de execução.

O *kernel* monolítico do sistema operacional Linux (Maxwell, 2000), contudo, possui um subsistema de gerência de módulos de *kernel*, que inclui funções de inserção, remoção, verificação da necessidade e teste de utilização para os módulos. Este subsistema foi projetado inicialmente para a implementação de drivers de dispositivos e a conseqüente redução do tamanho do *kernel*. Vários trabalhos na área de sistemas operacionais utilizaram esta funcionalidade para realizar a configuração de partes internas do *kernel*, como o escalonamento de processos (Barabanov, 1997). Nota-se que, por não ser essa a finalidade prevista para esse subsistema e por não ser a adaptabilidade uma filosofia de projeto do Linux, várias são as modificações que devem ser feitas no próprio *kernel* para que ele aceite e utilize uma nova estratégia de escalonamento de processos, por exemplo.

De uma forma geral, o *kernel* do Linux deve ser modificado para tornar adaptáveis certos mecanismos cuja configuração dinâmica a partir de novos algoritmos é interessante. Abrir essas "brechas" no *kernel* traz a necessidade de políticas de segurança capazes de impedir que a introdução de um novo módulo prejudique o funcionamento do sistema. A própria estrutura programada para receber os novos algoritmos pode prover alguma proteção ao sistema, como ocorre com a estrutura hierárquica de escalonamento ao garantir o isolamento entre as categorias de aplicações. Mas muitas são as alternativas para que um usuário mal intencionado promova um congelamento do sistema ou quebre seu sistema de segurança. A discussão sobre esse assunto desvia-se dos objetivos principais do presente trabalho e, portanto, não será aqui aprofundada.

Exemplos de módulos interessantes para serem inseridos em um sistema em tempo de execução para a criação de um novo serviço de provisão de QoS são: estratégias de admissão de processos e de fluxos de rede; estratégias de escalonamento de processos e de pacotes; estratégias de monitoração da carga imposta a um certo recurso; estratégias de classificação de fluxos de pacotes, entre outros.

## 2.7 Resumo do Capítulo

O presente Capítulo apresentou uma discussão sobre os subsistemas de processamento e comunicação dos GPOS, ressaltando as deficiências que impedem a predição do seu comportamento mediante as diferentes situações de carga e necessidades das aplicações. Em seguida, foi proposto um modelo simplificado que abrange de forma genérica os recursos de processamento e comunicação envolvidos na execução de aplicações multimídia distribuídas. Com esse modelo, a dependência existente entre os fluxos de dados e de instruções foi evidenciada, o que levou à conclusão de que existe a necessidade de orquestração de recursos internamente ao sistema operacional, entre CPU e buffers de comunicação. A partir das diferentes formas de implementação da pilha de protocolos, o modelo mostrou que cada um dos processos (sejam de aplicações, sejam de entidades de protocolo) e cada uma das filas de comunicação devem ter garantidas suas necessidades de uso de tais recursos. Tornou-se imperativo, portanto, que essa abordagem fosse considerada na construção da arquitetura.

Vista a necessidade de provisão de QoS em sistemas operacionais, vários trabalhos sobre o assunto foram descritos. Além de expor as soluções propostas para cada um dos pontos críticos, o texto deste Capítulo procurou identificar as estruturas e mecanismos definidos em comum, entre as diversas implementações. Foram vistas alternativas para o escalonamento de recursos, entidades de controle de admissão, gerenciadores de recursos, estratégias de mapeamento, formas de adaptação de serviços em sistemas operacionais, métodos para a solicitação de serviços e estabelecimento de contratos, entre outros. Esse estudo forneceu o embasamento necessário para a definição dos elementos que compõem a arquitetura proposta no Capítulo 3.