3
Uma Proposta para Reserva de Recursos no Nível do Usuário

Dado que nenhuma das ferramentas de reserva de recursos no nível do usuário analisadas durante a fase inicial de nossos estudos proviam as características de gerenciamento que gostaríamos de investigar, decidimos implementar uma ferramenta própria que provesse, além do controle de consumo de múltiplos recursos computacionais, facilidades de uso, de implementação e de extensão de suas regras de escalonamento. Este capítulo apresenta o resultado final dessa implementação, descrevendo como os componentes de provisão de reserva de recursos foram implementados e quais fatores foram decisivos na escolha das técnicas utilizadas. As descrições a serem apresentadas são de grande importância para o entendimento das consequências de se implementar mecanismos de reserva fora do núcleo do SO, pois focam nas funcionalidades e vantagens do conjunto de ferramentas de reserva proposto, em suas limitações e no porquê dessas limitações existirem.

3.1
Arquiteturas de Gerenciamento e a Reserva de Recursos

Em Sistemas Operacionais de Propósito Geral, onde os objetivos de gerenciamento são direcionados a satisfazer objetivos globais e não as metas diferenciadas das aplicações, arquiteturas de *middleware* podem ser utilizadas para agregar funcionalidades capazes de gerenciar consistentemente os recursos computacionais, de diminuir as complexidades de desenvolvimento de novas aplicações e de, consequentemente, reduzir os custos do desenvolvimento. Arquiteturas de *middleware* para o gerenciamento de recursos encontradas na literatura tendem a apresentar componentes básicos em suas formações [56, 57, 58, 59, 60, 61]. Nesses trabalhos é comum, por exemplo, a existência de módulos para a extração do perfil de consumo de recursos das aplicações, a tradução de requisitos de QoS de alto nível em reservas de recursos, o controle de admissão de novas aplicações e a negociação de QoS a ser proporcionada, além de módulos para a realização do escalonamento, da execução e da monitoração das aplicações.

A figura 3.1 apresenta a proposta de uma arquitetura de gerenciamento

PUC-Rio - Certificação Digital Nº 0521501/CA

de recursos que vamos adotar como referência neste trabalho. Apesar de arquiteturas mais genéricas do que a apresentada existirem, optamos por uma visão da atividade de gerenciamento mais particularizada, focada na provisão de qualidade de serviço através de reservas de recursos computacionais. A análise das interações entre os componentes dessa arquitetura pode nos auxiliar na identificação de pontos de apoio à atividade de provisão de QoS, contextualizando os mecanismos de reserva de recursos nessa arquitetura de gerenciamento.

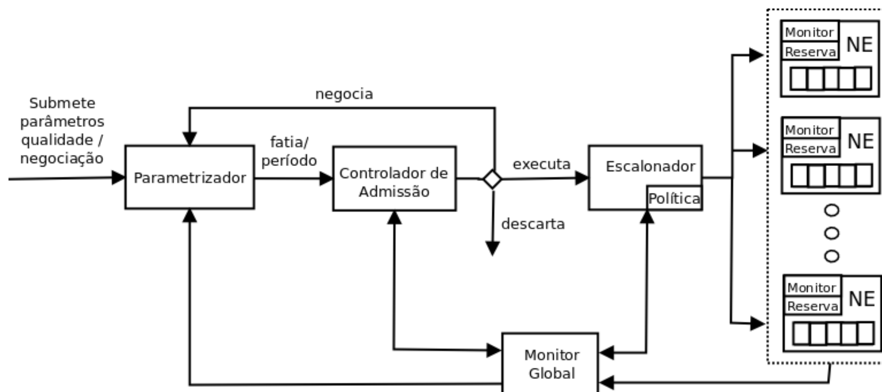

Figura 3.1: Arquitetura de gerenciamento de recursos.

Em nossa arquitetura, os nós de execução (NE) se registram junto a um serviço de monitoramento global que mantém uma lista dos nós ativos e dos estados desses nós. As informações coletadas pelo monitor são utilizadas pelo componente parametrizador, pelo controlador de admissão e pelo escalonador, pois é a partir delas que essas entidades determinam, por exemplo, características de execução das aplicações, a disponibilidade das máquinas executoras e momentos em que adaptações na qualidade de serviço provida são necessárias.

Ao submeter uma aplicação, o usuário também deve submeter os parâmetros de qualidade desejados e o tipo de negociação a ser realizada no caso de não haver recursos suficientes para atender à requisição de maneira satisfatória. De posse desses parâmetros, é papel do parametrizador determinar qual deve ser o montante de recursos computacionais necessários para se atingir a qualidade esperada. Após ser estimado pelo parametrizador, o valor do montante é repassado juntamente com um pedido de execução ao escalonador do sistema.

Ao receber uma requisição de execução, o escalonador, orientado por uma política de escalonamento, deve escolher, entre os nós de execução disponíveis, aqueles que apresentam as melhores condições de executar os diferentes componentes da aplicação. Em seguida, os comandos para início

PUC-Rio - Certificação Digital Nº 0521501/CA

de execução dos componentes são enviados para os nós escolhidos, onde mecanismos de reserva, de monitoração e de adaptação garantem a qualidade de serviço esperada para cada tipo de recurso especificado.

Apesar da provisão de QoS depender do bom funcionamento da arquitetura de gerenciamento como um todo, a reserva de recursos está mais intimamente ligada aos nós de execução, pois é nesses componentes que os mecanismos de reserva residem. Por esse motivo, é nesse componente que este trabalho se concentra.

Nos nós de execução, as ferramentas de reserva podem limitar e garantir o uso de recursos locais através da moldagem de tráfego e reordenação de requisições de acesso aos dispositivos computacionais. Além disso, por acompanharem de maneira muito precisa o uso de recursos por um processo, ferramentas de reserva podem prover esse tipo de informação para outros serviços presentes nos nós execução como, por exemplo, um serviço responsável por determinar quando adaptações nos valores de reserva garantidos necessitam ser realizadas.

3.2 O Conjunto de Ferramentas ReservationSuite

Visando não somente o controle do consumo de recursos e uma baixa sobrecarga do sistema, mas também as facilidades de uso, implementação e extensão, três ferramentas de reserva foram implementadas inteiramente no nível do usuário, isto é, sem que alterações no núcleo do sistema operacional fossem necessárias. Essa abordagem de gerenciamento de recursos no nível do usuário foi baseada no DSRT [18], um sistema para o gerenciamento de CPU baseado em reservas. Porém, algumas funcionalidades foram inseridas nessa ferramenta a fim de torná-la mais adequada na tarefa de isolar o desempenho de aplicações. A tabela 3.1 apresenta a síntese das principais diferenças entre essas duas ferramentas.

Tabela 3.1: Funcionalidades presentes no ReservationSuite e no DSRT.

Funcionalidade	ReservationSuite	DSRT
Controle de CPU	SIM	SIM
Controle de E/S	SIM	NÃO
Flexibilidade no escalonamento	SIM	NÃO
API	NÃO	SIM
Controle de processadores	SIM	NÃO
Adaptação	SIM	SIM
Caracterização das aplicações	NÃO	SIM

PUC-Rio - Certificação Digital Nº 0521501/CA

As características de controle de E/S de dados, flexibilidade no escalonamento de processos e controle de processadores, presentes somente no ReservationSuite foram incorporados à nossa ferramenta com o objetivo de proporcionarem mais autonomia no gerenciamento dos recursos. Já a interface de programação e a caracterização das aplicações, funcionalidades existentes somente no DSRT, não foram implementadas no ReservationSuite, mas, devido aos benefícios que essas funcionalidades podem trazer para o melhor aproveitamento dos recursos físicos das máquinas, devem ser futuramente inseridas em nossa ferramenta.

Os mecanismos de reserva implementados no ReservationSuite englobam as atividades de limitação e de garantia de acesso aos recursos. A limitação de acesso assegura que as aplicações não irão utilizar mais recursos do que o acordado no momento de sua admissão, mesmo em situações em que existam recursos ociosos no sistema. Assim, o uso de um recurso por uma aplicação tem que ser necessariamente menor ou igual à reserva destinada a essa aplicação. Ou seja,

$$U_i \leq R_i$$

onde U_i e R_i representam, respectivamente, a utilização corrente e a reserva do recurso i requisitado por uma determinada aplicação.

A eficácia da limitação de uso é importante, pois assegura que a fatia não reservada do recurso sempre estará disponível para novas reservas. Essa característica de disponibilidade aliada a mecanismos de priorização de uso dos recursos e de controle de admissão de novas reservas garantem que as taxas de acesso das aplicações sejam atendidas sempre que as cargas de trabalho geradas pelas aplicações necessitarem.

Duas das ferramentas implementadas apresentam mecanismos para garantir e limitar o uso das larguras de banda do processador e do disco, enquanto uma terceira apresenta facilidades para gerenciar o acesso à rede de comunicação. Esses recursos foram escolhidos porque são intensamente utilizados por diferentes nichos de aplicações e, assim, frequentemente, representam gargalos de desempenho que dificultam a provisão de qualidade de serviço. Por razões de limitação de escopo, a reserva de memória não foi implementada, mas diferentes mecanismos para a limitação do uso de memória foram investigados e constatou-se que eles podem ser inseridos na ferramenta, porém, com determinadas limitações decorrentes de conflitos com reservas de largura de banda de disco e do uso de bibliotecas compartilhadas.

Para facilitar o uso concomitante de diferentes tipos de reserva, as três ferramentas foram agrupadas em uma única ferramenta chamada Reservation-

PUC-Rio - Certificação Digital Nº 0521501/CA

Suite. O ReservationSuite segue uma arquitetura cliente-servidor. Nessa arquitetura, a comunicação entre os processos é feita por meio de *sockets*, fato que facilita o uso do conjunto de ferramentas em infraestruturas computacionais distribuídas.

Na suíte de ferramentas, o processo servidor é o responsável por carregar as propriedades do sistema, esperar por requisições de reservas e de adaptação e realizar um certo controle de admissão. Também são tarefas do processo servidor monitorar o consumo de processamento das aplicações e mapear regiões de memória a serem utilizadas pelas funções de interposição para o controle de acesso ao disco. Devido às duas últimas tarefas, o servidor do ReservationSuite deve ser executado na mesma máquina onde as reservas devem ser garantidas.

As propriedades do ReservationSuite são carregadas a partir de um arquivo de configuração ou configuradas diretamente na linha de comando durante a inicialização do servidor. No arquivo de propriedades devem ser especificados o limite de processamento a que o sistema operacional é subordinado (informação necessária em domínios virtualizados, por exemplo), os valores que representam os pontos de saturação do disco e da rede de comunicação (utilizados no controle de admissão de novas reservas) e o caminho das bibliotecas compartilhadas que implementam as funções de controle de E/S e o escalonamento de processos.

Durante a inicialização do servidor, deve-se especificar na linha de comando a porta TCP onde o servidor aguarda por novas conexões e uma máscara de *bits* que informa quais processadores da máquina podem ser utilizados nas reservas. Um parâmetro opcional na linha de comando é a definição da política de escalonamento a ser utilizada. Caso essa opção não seja explicitamente configurada, busca-se ativar a política implementada pela biblioteca compartilhada cujo caminho é indicado no arquivo de propriedades do servidor. Caso a propriedade correspondente não apresente um caminho válido, uma política padrão embutida no código do servidor é ativada.

A linha de comando a seguir ilustra um caso onde o servidor é iniciado na porta 8000, com dois dos processadores da máquina local ativados (máscara de *bits* igual a 3) e utilizando a política de escalonamento *earliest deadline first*.

```
./server -p 8000 -m 3 -f edf.lua
```

O processo cliente é o responsável por requisitar a execução de aplicações ao servidor e especificar quais são os montantes de reservas de recursos necessários para que a aplicação seja executada com um determinado desempenho. Dessa maneira, para submeter uma aplicação para execução é necessário

PUC-Rio - Certificação Digital Nº 0521501/CA

especificar o nome e a porta TCP do servidor de reservas, a linha de comando para iniciar a aplicação e as taxas de processamento requeridas a cada período de tempo informado. Parâmetros opcionais especificam se a reserva de processamento deve ou não ser expandida em caso de processamento ocioso no sistema (propriedade chamada de *work-conserving*), além de reservas de disco e de rede.

Como exemplo, a linha de comando a seguir apresenta uma submissão da aplicação `iozone` iniciada com o parâmetro "`-a`" sendo submetida à execução no servidor *node01*, na porta 8000, com uma reserva de processamento de 33% (500 ms a cada 1500 ms) e uma taxa de leitura/escrita em disco de 4MB/seg (4194304 *bytes* a cada 1000 ms).

```
./client -m node01:8000 -P 1500 -p 500 -D 1000 -d 4194304 iozone -a
```

No projeto da ferramenta, o formato das reservas de recursos — com um intervalo de tempo seguido de uma quantidade de acesso (-P e -p, -D e -d, -N e -n) — poderia ser substituído, por exemplo, por um formato em que se expressasse somente a quantidade absoluta de segundos de acesso à CPU ou de *bytes* lidos/escritos na rede ou no disco. A primeira opção foi escolhida porque é semelhante à forma com que o sistema operacional calcula as frações de recursos utilizadas. Assim, ela facilita o compartilhamento simultâneo de recursos. Além disso, tratar o uso dos recursos de forma cíclica permite que aplicações com requisitos de desempenho dados por intervalo de tempo sejam beneficiadas.

Se a requisição de execução do cliente for aceita, a aplicação será iniciada com o montante de recursos especificado na requisição. A alteração dos parâmetros de reserva pode ser feita através de um comando do tipo:

```
./adapt -m node01:8000 -P 1000 -p 500 -D 1000 -d 2097152 <pid>
```

onde `pid` é o identificador do processo da aplicação sendo executada pelo servidor. Nesse caso, a porcentagem de reserva de processamento foi alterada para 50% e a taxa de leitura/escrita em disco, alterada para 2MB/seg.

Pedidos de reserva e de adaptação aceitos cobrem toda a árvore de processos de uma aplicação. Assim, à medida que processos são criados a partir de uma mesma aplicação, eles compartilham o montante de recursos alocados ao processo pai no momento em que a aplicação é iniciada. Essa prática evita que processos filhos burlem os acordos de uso de recursos estabelecidos.

A arquitetura do ReservationSuite e a interação entre seus componentes são apresentadas na figura 3.2. Essa arquitetura é composta principalmente por *threads*: uma para esperar requisições de clientes, uma para tratar tais

PUC-Rio - Certificação Digital Nº 0521501/CA

requisições, uma para monitorar a árvore de processos e outra para tratar os alarmes que gerenciam os tempos de execução dos processos.

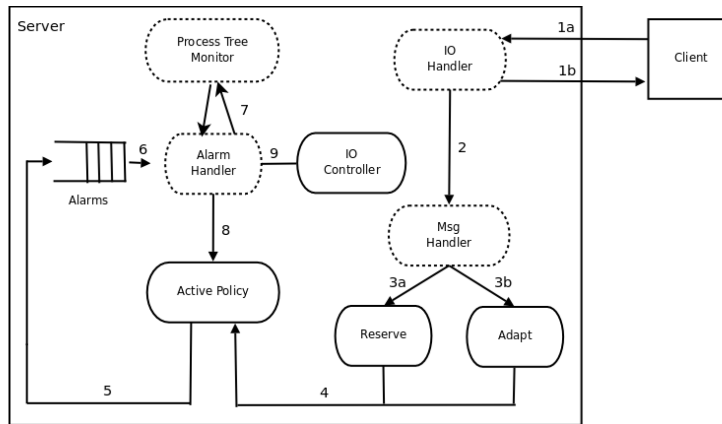

Figura 3.2: Arquitetura do ReservationSuite.

Ao receber uma mensagem do cliente através da *thread* IO-handler (1a), o servidor inicia uma nova *thread* para tratar da requisição (2). A *thread* iniciada trata pedidos de reserva e de adaptação de maneira diferenciada, mas, em ambos os casos, ela verifica se os parâmetros da mensagem são consistentes e se podem ser atendidos com a qualidade desejada (controle de admissão). No caso da reserva (3a), se o pedido puder ser atendido, o processo especificado é criado, a identificação do processo é retornada ao cliente (1b) e o seu tempo de processamento e a taxa de acesso ao disco/rede começam a ser monitorados. No caso da adaptação (3b), novos parâmetros de reserva são atualizados para o processo e uma mensagem de sucesso é enviada para o cliente (1b).

Após tratar uma requisição, o servidor invoca a política de escalonamento ativa (4) para configurar novos alarmes (5). O controle de reserva de processamento é feito por meio de alarmes que expiram após uma fatia de uso da CPU (6). Os alarmes são ordenados de acordo com a política de escalonamento carregada durante a inicialização do servidor. As *threads* de monitoração e de tratamento de alarme trocam informações quando a última precisa obter o consumo de CPU de todos os processos criados a partir do processo monitorado (7). Chamadas à política de escalonamento também são feitas após o término de tratamento de um alarme (8).

Se um processo é iniciado com reservas de acesso ao disco e à rede, ele é carregado juntamente com uma biblioteca de controle de E/S a qual é responsável por limitar e garantir aos processos o acesso às mídias de disco e à interface de rede. Essa biblioteca monitora as atividades de E/S de dados realizadas pelo processo e regula as taxas de E/S realizadas por ele inserindo

PUC-Rio - Certificação Digital Nº 0521501/CA

PUC-Rio - Certificação Digital Nº 0521501/CA

retardos na sua execução (9). O monitoramento de E/S é realizado através da leitura regular de blocos de memória compartilhados entre o servidor ReservationSuite e as funções de E/S implementadas na biblioteca.

Em um ambiente de computação compartilhado, a facilidade de implementar novas políticas, assim como a possibilidade de escolher as políticas de escalonamento ativas, ou mesmo combiná-las, são características bastante interessantes, visto que para satisfazer as necessidades específicas de cada aplicação, uma política pode ser mais apropriada que outra. No entanto, essas facilidades são pouco exploradas, seja em ferramentas no nível do usuário, seja em sistemas operacionais de propósito geral. Tipicamente, nesses sistemas, um desenvolvedor precisa ter um profundo conhecimento sobre as estruturas de dados utilizadas no escalonamento e sobre como os processos são tratados para, então, implementar novas políticas. Após a implementação, ainda é necessário compilar os arquivos fontes para tornar a implementação efetiva. Essa prática de alterar um código fonte e recompilá-lo tende a gerar bom desempenho, mas pode ser complexa ou mesmo inviável no caso de *softwares* sem código livre. Por esse motivo, decidimos adotar uma prática diferente no ReservationSuite: em nossa suíte de ferramentas, políticas de escalonamento podem ser facilmente implementadas e carregadas.

A facilidade de implementação de políticas no ReservationSuite ocorre porque as mesmas podem ser desenvolvidas na linguagem de *script* Lua ou em C. Em ambos os casos, o desenvolvedor manipula apenas as variáveis que influenciam as regras de escalonamento, sem que haja preocupações com a proteção de seções críticas, a configuração de alarmes ou a interação com outros componentes da ferramenta. No caso das políticas escritas em Lua, sequer as estruturas de dados utilizadas pelo servidor ReservationSuite para representar um processo precisam ser conhecidas pelo desenvolvedor.

Os dados de um processo aos quais uma política de escalonamento do ReservationSuite tem acesso são descritas na tabela 3.2. Nessa tabela, a primeira coluna apresenta os nomes das propriedades do processo, enquanto a segunda e terceira coluna apresentam os valores iniciais dessas propriedades e suas descrições, respectivamente. Esses dados são suficientes para implementar algumas políticas de escalonamento envolvendo tanto o controle de processamento como também o controle de entrada/saída de *bytes* no disco e na interface de rede.

No ReservationSuite, quando o desempenho é mais importante do que a facilidade de se programar no ambiente Lua, novas políticas podem ser implementadas como bibliotecas compartilhadas C. Nesse caso, a biblioteca contendo a política é dinamicamente carregada junto com o processo servidor

Tabela 3.2: Propriedades dos processos no ReservationSuite.

Propriedade	Valor Inicial	Descrição
period	Especificado na submissão	Intervalo para executar uma fatia de tempo
diskPeriod	Especificado na submissão	Intervalo para acessar o disco
netPeriod	Especificado na submissão	Intervalo para acessar a rede
slice	Especificado na submissão	Tempo que o processo deve executar a cada período
diskSlice	Especificado na submissão	Montante de bytes que o processo deve ler/escrever a cada *diskPeriod*
netSlice	Especificado na submissão	Montante de bytes que o processo deve enviar a cada *netPeriod*
executedTime	0	Tempo de execução acumulado
onPeriodExecutedTime	0	Tempo de execução acumulado no período corrente
onPeriodDiskAmt	0	Montante de bytes lidos/escritos no período corrente
onPeriodNetAmt	0	Montante de bytes enviados no período corrente
alarmTime	Hora de criação do processo	Próximo alarme do processo a ser tratado
conserving	0	Informa se o proc. está consumindo processamento extra
isConserving	Especificado na submissão	Informa se o proc. deve consumir mais CPU em caso de processamento ocioso
isRunning	0	Informa se o proc. está sendo executado
numChild	1	Número de processos filhos do proc. incluindo ele mesmo

no momento de sua inicialização. Para isso, o processo servidor é invocado a partir de um envoltório (do termo inglês *wrapper*) e utilizando a variável de

PUC-Rio - Certificação Digital Nº 0521501/CA

ambiente LD_PRELOAD a qual sobrescreve a função do servidor responsável por implementar a política de escalonamento. No caso de uma política Lua, o arquivo onde a política é implementada deve ser especificado na linha de comando do servidor. Se nenhuma das opções descritas puder ser utilizada, uma política padrão que prioriza processos cujos alarmes expiram mais cedo, implementada em C e embutida no código do servidor, é ativada.

As próximas seções descrevem, em detalhes, como cada uma das três ferramentas de reserva contidas no ReservationSuite foram implementadas. O entendimento dessas seções é essencial para a identificação de quais são as vantagens e desvantagens do uso de mecanismos de controle de recursos no nível do usuário e de como elas podem interferir no uso da ferramenta em determinados contextos.

3.2.1 Reserva de Processamento

A reserva de processamento no ReservationSuite é realizada pela ferramenta CPUReserve [24] a qual, de acordo com uma política de escalonamento, gerencia a reserva de processamento através de chamadas ao sistema (*system calls*) que dinamicamente alteram as prioridades dos processos. Como a manipulação de prioridades limita o espaço de decisão do escalonador do sistema operacional a um subgrupo de processos, ela faz com que os processos consumam mais ou menos processamento em um determinado período.

CPUReserve é implementado utilizando-se a API Linux[1]. Quando o servidor é iniciado, sua execução é limitada a um conjunto de processadores da máquina local através da chamada ao sistema *sched_setaffinity*. A limitação de processadores atinge não somente a execução do servidor, mas também as dos processos que esse servidor gerencia.

No Linux, o acesso à CPU pode ser gerenciado por diferentes escalonadores. Por padrão, o escalonador ativado é o SCHED_OTHER, o qual implementa uma política de *round-robin* em que os processos tendem a ganhar acesso à CPU à medida que se mostram mais interativos. Nesse caso, não há atribuições estáticas de prioridades. O contrário ocorre com as políticas de tempo-real tal como a SCHED_RR. Essa política também consiste em uma implementação da política *round-robin*, mas atribui prioridades e fatias de tempo aos processos. Processos de maior prioridade são priorizados no acesso à CPU, sendo capazes de causar *preempção* em processos de menor prioridade.

Para gerenciar de maneira eficiente a requisição de processos clientes, o servidor é executado com a maior prioridade do sistema para processos de

[1] Há também uma versão para plataformas Windows que utiliza a API win32.

PUC-Rio - Certificação Digital Nº 0521501/CA

PUC-Rio - Certificação Digital Nº 0521501/CA

tempo-real. Para isso, são utilizadas as chamadas *sched_get_priority_max* e *sched_setscheduler* parametrizadas com a política de tempo real SCHED_RR. Enquanto a chamada *sched_get_priority_max* informa a maior prioridade disponível na política SCHED_RR, a chamada *sched_setscheduler* altera o escalonador para o SCHED_RR. Por esse motivo, o servidor deve ser executado com privilégios de administrador.

Quando um processo cliente começa a ser executado no CPUReserve, o servidor o transforma em um processo de tempo-real com a segunda maior prioridade do sistema. A alta prioridade dos processos garante que eles sempre terão acesso à CPU quando estiverem no estado pronto de execução.

Processos clientes são constantemente monitorados e, se eles excederem a fatia de tempo de execução determinada para o seu período, podem ser suspensos ou ter as suas prioridades reduzidas a um mínimo permitido para um processo comum (política SCHED_OTHER) até que o próximo período se inicie. A decisão de suspender ou reduzir a prioridade de um processo é baseada no parâmetro *work-conserving* informado pelo processo cliente no momento do seu pedido de reserva ou adaptação.

Para limitar o conjunto de processadores que podem ser utilizados pelos processos gerenciados pelo ReservationSuite, a chamada *sched_setaffinity* é utilizada. Em seguida, o processo é iniciado com a chamada *execvp*.

A limitação do uso de processamento é realizado através de alarmes de tempo-real que, quando expiram, invocam uma *thread* que decide se um processo deve continuar a sua execução, ser suspenso ou ter a sua prioridade alterada. A figura 3.3 ilustra um diagrama de estados de um processo e as decisões tomadas pela *thread* a cada vez que um alarme expira.

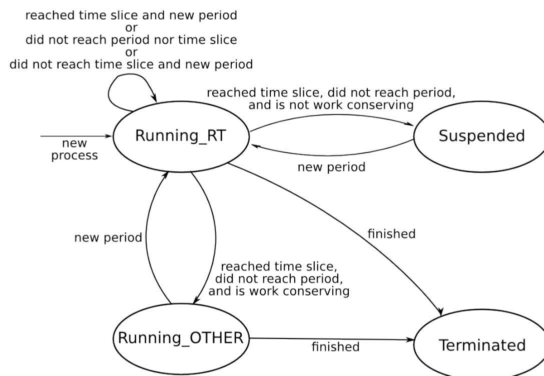

Figura 3.3: Diagrama de caminhos entre os estados de um processo.

Quando um alarme expira, a *thread* do servidor verifica o estado do processo que requisitou o alarme e manipula as suas propriedades (tabela 3.2). Se o estado do processo for:

1. Terminado: o processo é removido da tabela de processos do servidor;

2. Suspenso: significa que no período anterior, o processo executou sua fatia de tempo, foi suspenso até o final do seu período e agora deve executar um novo ciclo de processamento, isto é, iniciar uma nova fatia de tempo de execução em um novo período. Nesse caso, a propriedade *isRunning* do processo é configurada para 1, *nextPeriod* é atualizado para o próximo período, *onPeriodExecutedTime* é configurado para 0 e *nextAlarm* é configurado para os próximos *slice* milissegundos;

3. Executando como um processo comum: significa que o alarme expirou porque o processo atingiu o fim de seu período. Assim, o processo deve ser transformado em um processo de tempo-real e iniciar uma nova fatia de tempo de execução. Nesse caso, a propriedade *isConserving* é configurada para 0, *nextPeriod* é atualizado para o próximo período, *onPeriodExecutedTime* é configurado para 0, *executedTime* é atualizado e *nextAlarm* é configurado para os próximos *slice* milissegundos;

4. Executando como um processo de tempo-real: nesse caso, se a fatia de tempo de execução do processo para o período corrente:

   (a) Foi executada:

   - Se o período do processo:
     - Expirou: o processo pode executar novamente por uma outra fatia de tempo. Nesse caso, a propriedade *nextPeriod* é atualizada para o próximo período, *onPeriodExecutedTime* é configurada para 0 e *nextAlarm* é configurada para os próximos *slice* milissegundos;
     - Não expirou: se o processo é do tipo *work-conserving*, ele é transformado em um processo comum de baixa prioridade e habilitado a executar até o final do período. Nesse caso, a propriedade *isConserving* é configurada para 1 e *nextAlarm* é configurada para o tempo restante até *nextPeriod*. Se o processo não for *work-conserving*, ele é suspenso até o próximo período. Nesse caso, a propriedade *isRunning* é configurada para 0, *executedTime* é atualizada e *nextAlarm* é configurado para o tempo restante até *nextPeriod*.

   (b) Não foi executada:

   - Se o período do processo:

PUC-Rio - Certificação Digital Nº 0521501/CA

- Expirou: significa que houve um erro. Esse erro pode indicar que há mais reservas do que os processadores disponíveis podem tratar ou simplesmente que o processo não precisou ocupar toda a fatia de processamento reservada a ele naquele período. Para evitar a parada do servidor, o processo é habilitado a continuar a executar, reiniciando a sua fatia de tempo. Assim, a propriedade *nextPeriod* é atualizada para o próximo período, *executedTime* é atualizada, *onPeriodExecutedTime* é configurada para 0 e *nextAlarm* é configurada para os próximos *slice* milissegundos;
- Não expirou: se ainda for possível executar o que falta da fatia de tempo antes de um novo período ser iniciado, o processo é habilitado a executar pelo tempo restante da fatia de tempo. Nesse caso, a propriedade *nextAlarm* é configurada para o tempo restante da fatia de tempo. Caso contrário, um erro similar ao anteriormente descrito ocorreu e, assim, *nextAlarm* é configurada para o tempo restante do período atual.

PUC-Rio - Certificação Digital Nº 0521501/CA

3.2.2
Reserva de Largura de Banda de Disco

Um dos principais objetivos de escalonadores de E/S consiste em otimizar o tempo de acesso a disco rígidos, os quais são considerados mídias de alto custo de acesso. Exemplos de otimização incluem as técnicas de aglutinação (*merging*) e de elevação (*elevator*). Na primeira técnica, o escalonador de E/S agrupa requisições de acesso a áreas próximas do disco a fim de reduzir o deslocamento da cabeça do disco (*disk seeking*); na segunda, o escalonador ordena requisições baseado nas localizações físicas relativas aos blocos do disco em uma tentativa de prosseguir o acesso em uma mesma direção o máximo possível. Essas técnicas, aliadas a otimizações de *cache*, dificultam a implementação de uma ferramenta de reserva de largura de banda de disco no nível do usuário porque, muitas vezes, uma chamada ao sistema para entrada/saída de dados não gera uma entrada/saída no disco de fato [62, 52].

Felizmente, existem meios providos pelos próprios sistemas operacionais que podem facilitar o controle de E/S. Considerando o sistema operacional Linux, a opção mais simples é a interposição de funções relacionadas a entrada e saída de dados do disco. Com a técnica de interposição, uma biblioteca compartilhada é utilizada para, em tempo de execução, sobrescrever funções tais como *read* e *write*. Por esse motivo, essa técnica não pode ser utilizada para

aplicações que sejam ligadas estaticamente. Nessas situações, pode-se optar pela técnica de interceptação de chamadas ao sistema (*system calls*), em que um processo pai é capaz de monitorar e controlar a execução de um outro processo, podendo examinar e alterar a imagem e os registradores desse processo. Porém, o uso de interceptação de código deve ser utilizado com cautela, uma vez que acarreta uma alta sobrecarga no processo sendo monitorado, assim como será apresentado nas próximas duas subseções e discutido na Seção 4.6. Dessa maneira, visando não comprometer o desempenho das aplicações executadas no ReservationSuite, resolvemos limitar o escopo da ferramenta de reserva de largura de banda de disco (assim como o da ferramenta de reserva de largura de banda de rede) a aplicações ligadas dinamicamente.

Interposição de Funções

No ReservationSuite, a limitação do uso de largura de banda de disco é realizada pela interposição de funções de acesso ao disco com assinaturas iguais às implementados pela `libC` e embutidas em uma biblioteca compartilhada. Para a substituição das funções utiliza-se a variável de ambiente LD_PRELOAD a qual instrui o ligador do SO a carregar a nossa biblioteca juntamente com aquelas que foram especificadas quando os programas a serem gerenciados foram compilados. LD_PRELOAD é configurada no momento de criação do processo através da função *putenv*. Em seguida, o comando *ionice* é invocado para alterar a prioridade de acesso ao disco desse processo.

O controle de E/S implementado pela biblioteca compartilhada é realizado com o suporte de regiões de memória mapeadas. Para realizar esse mapeamento, são invocadas as chamadas *shmget* e *shmat*, responsáveis por alocar e compartilhar um segmento de memória, respectivamente.

Para limitar o acesso ao disco, foram sobrescritas as funções *read*, *write* e afins como, por exemplo, *readv/writev*, *pread/pwrite* e *fprintf*. Na sobrescrita das funções a região de memória compartilhada com o servidor informa o quanto o processo pode escrever em um determinado período de tempo e o quanto ele já utilizou desse montante. Se toda a taxa de leitura/escrita já tiver sido utilizada em um mesmo período de tempo, então o processo se torna inativo (dormindo) até o início de um novo período de tempo (novo ciclo). Ou seja, sempre que um processo acessar o disco mais do que o permitido para ele durante um determinado período, atrasos são inseridos na sua execução de modo que o consumo extra de largura de banda de disco não prejudique outros processos que precisam acessar a mesma mídia.

Para garantir o acesso ao disco, há um controle do montante de largura de banda de disco requisitado por todos os processos clientes sendo gerenciados

PUC-Rio - Certificação Digital Nº 0521501/CA

pelo servidor. Enquanto a soma de todas as requisições for menor que a taxa de saturação do disco, novos processos podem ser admitidos. Nesse ponto, são identificados dois problemas: como estimar a taxa de saturação de disco e como garantir que um processo externo ao ReservationSuite não interfira na taxa de serviço do disco sendo considerado. Enquanto a estimativa da taxa de saturação de disco pode ser resolvida com o uso de *benchmarks*, o isolamento de desempenho do disco pode ser apenas amenizado com o uso de priorização de acesso ao disco.

O conceito de priorização de acesso ao disco foi introduzido no núcleo do Linux juntamente com o escalonador de E/S chamado CFQ (*Completely Fair Queueing*). Como consequência, os mantenedores do Linux introduziram o comando *ionice* no SO como um meio de permitir que administradores mudassem a classe de escalonamento de E/S de processos, assim como o comando *nice* faz para o processamento. Com o uso do comando *ionice*, processos com maior prioridade são favorecidos no acesso à mídia de disco ao requisitar operações de leitura e escritas síncronas (uso de *flags* O_DIRECT ou O_SYNC). Operações assíncronas de escrita não são tratadas pelo mecanismo de priorização[2].

A figura 3.4 apresenta um pseudocódigo para a sobrescrita da função *write* presente na `libC`. Com a sobrescrita dessa função, toda vez que a operação de *write* for invocada na aplicação sendo gerenciada pelo servidor ReservationSuite, o ponteiro para a implementação dessa operação será desviado da `libC` para a biblioteca que implementa a nova função *write*. De acordo com a figura 3.4, a aplicação (à esquerda) tenta escrever um bloco de 2048 *bytes* de uma única vez. Supondo que a reserva de acesso ao disco dessa aplicação seja de 1024 *bytes* por segundo, o código interposto divide o pedido em duas requisições de 1024 *bytes* cada e insere um retardo de aproximadamente 1 segundo (1 segundo menos o tempo gasto pela ferramenta para o controle de *bytes* escritos e para a operação de escrita em si) após cada requisição de escrita. Chamadas ao sistema *write* são seguidas de uma operação *fflush* a qual força o envio de todos os dados dos *buffers* no espaço de usuário para os *buffers* do núcleo do SO. Depois que todos os *bytes* são enviados para escrita em disco, o controle é retomado pela instrução seguinte ao *write* que foi substituído.

Interceptação de Chamadas ao Sistema

A interceptação de chamadas ao sistema pode representar uma solução para aplicações ligadas estaticamente e que, por esse motivo, não podem carregar novas definições de funções com a ajuda da interposição. Em ambientes

[2]Conforme descrito no manual da operação ioprio_set.

PUC-Rio - Certificação Digital Nº 0521501/CA

```
...
char buffer[2048];
len = sizeof(buffer);
f = open(...);
write(f, buffer, len);
close(f);
...
```

```
int j = 0;
while(buffer+(j*1024) < len) {
   write(f, buffer+(j*len), 1024);
   fflush(fp);
   sleep(1);
   j++;
}
int r = len - (j*1024);
if(r > 0) {
   write(f, buffer+(j*1024), r);
   fflush(fp);
}
```

Figura 3.4: Exemplo de sobrescrita da operação de escrita em disco *write*.

Linux, a interceptação de código pode ser feita com o uso da chamada *ptrace*. Com o *ptrace* um processo pai pode observar e controlar a execução de seus processos filhos, além de examinar e alterar a imagem desses processos.

A principal desvantagem do uso do *ptrace* está na alta sobrecarga causada pelas sucessivas trocas de contexto necessárias durante a interceptação de uma chamada ao sistema. A cada interceptação de chamada ao sistema, são realizadas, no mínimo, 6 trocas de contexto entre processos no nível do usuário e processos no nível do núcleo do SO. A figura 3.5 ilustra os passos de uma interceptação de chamada ao sistema em uma aplicação composta de três processos os quais são monitorados por um processo externo via *ptrace*.

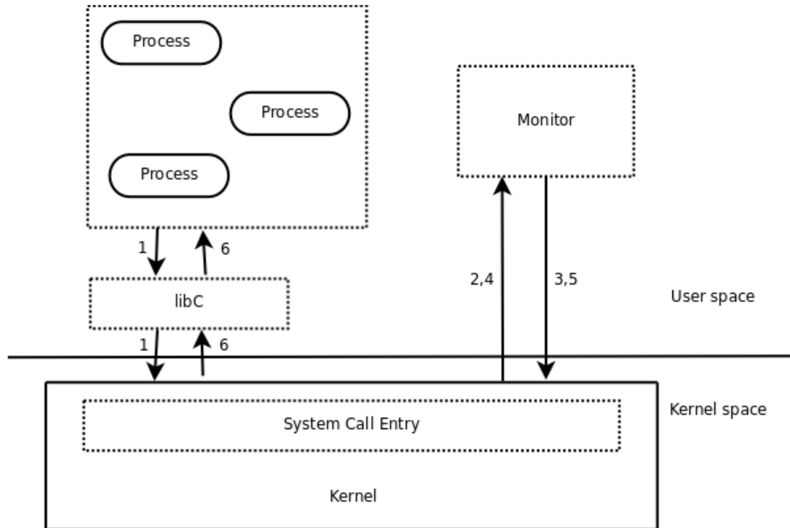

Figura 3.5: Etapas de uma interceptação de chamada ao sistema.

Primeiramente (1), um dos processos da aplicação sendo monitorada faz uma chamada ao sistema. Essa chamada passa pela `libC` e tem a sua execução desviada para o núcleo (*kernel*). Em seguida, o núcleo verifica que esse tipo de chamada está sendo acompanhada por um processo através do

PUC-Rio - Certificação Digital Nº 0521501/CA

ptrace e, assim, repassa a execução para o processo monitor (2) o qual examina e realiza possíveis alterações nos valores dos registradores utilizados pelo processo monitorado e retorna o controle para o núcleo executar a chamada(3). O núcleo, por sua vez, executa a chamada ao sistema e transfere o controle de volta ao monitor para que ele possa examinar o valor de retorno da chamada (4). Mais uma vez, o monitor examina/altera os valores dos registradores e retorna o controle ao núcleo (5). Finalmente, o núcleo transfere o controle ao processo monitorado.

Uma maneira de contornar a sobrecarga causada pelas trocas de contexto do *ptrace* é utilizar essa ferramenta apenas para alterar a imagem do processo sendo gerenciado, adicionando funcionalidades de controle de E/S a esse processo. A técnica consiste em inserir pedaços de código que controlam o acesso ao disco em algum lugar da pilha de dados do processo e, em seguida, alterar o ponteiro de instruções do processo para a localização do novo código. Contudo, nesse caso, os códigos a serem inseridos na imagem do processo devem ser implementados em linguagem de máquina, um procedimento difícil de ser feito, principalmente de forma dinâmica. Muitas ferramentas de instrumentação dinâmica de código, tais como a Dyninst [63] e Pin [64], foram investigadas, mas não encontramos nenhuma que pudesse ser utilizada em nossa plataforma de testes sem que fosse necessário recompilar o núcleo dos sistemas operacionais de nossas máquinas, uma tarefa que, para assegurar compatibilidade de versões de SO, não foi realizada. Sendo assim, somente para fins de comparação com a técnica de interposição, um protótipo de reserva de largura de banda de disco foi desenvolvido com a técnica de interceptação utilizando o *ptrace* sem reposição de ponteiros. Os resultados da comparação são apresentados na seção 4.6.

PUC-Rio - Certificação Digital Nº 0521501/CA

Saturação de E/S de Disco

Toda arquitetura de gerenciamento de recursos que proveja mecanismos de garantia de execução deve conter módulos que realizam o controle de admissão de novas reservas. Tipicamente, reservas são aceitas até que o recurso requisitado esteja saturado, ou seja, até não poder mais atender a novas requisições sem que reservas já asseguradas sejam comprometidas. Supondo que S seja a taxa máxima que o recurso é capaz de atender, a soma de todas as reservas R_i para esse recurso deve ser dada por:

$$\sum_{i=0}^{N} R_i < S$$

Em sistemas operacionais de tempo real, onde o tempo de resposta a

um evento deve ser previamente definido, a saturação dos recursos é fielmente determinada e garantida.

O estabelecimento das taxas de saturação dos recursos em sistemas operacionais de tempo real garante um controle mais preciso das características temporais das aplicação, mas não necessariamente provê um alto desempenho desses sistemas. Por esse motivo, tipicamente, sistemas operacionais de tempo real apresentam desempenho inferior ao de sistemas operacionais de propósito geral [65].

Mas qual é a taxa de E/S necessária para saturar um disco? Para entender o conceito de saturação em disco é preciso entender que existem dois tipos de acesso ao disco: o sequencial e o aleatório. A velocidade dos acessos sequenciais é muito influenciada pela velocidade de transferência do disco, enquanto a velocidade de acessos aleatórios é muito influenciada pelo tempo de busca (*seek time*) do *driver* do disco. Utilizar o resultado de um *benchmark* com acessos sequenciais como ponto de saturação de disco pode ser irreal em casos onde executam-se diversas aplicações em paralelo. Isso porque a probabilidade de que elas acessem diferentes áreas do disco é grande. Há ainda que se considerar o tipo de operação realizada pelo *benchmark*: se é síncrona ou assíncrona e se utiliza ou não a *cache* de arquivos. Assim, em situações onde não se conhece o perfil de acesso ao disco das aplicações a serem gerenciadas, a execução de um *benchmark* com requisições aleatórias ao disco e sem o uso de *caches* de arquivos pode apresentar um limite superior (pior caso) de acesso ao disco. Esse limite pode ser utilizado como medida para estimar a saturação do disco e, assim, determinar o quanto aplicações podem requisitar de banda de disco. A ideia é que requisitando somente o que o disco é capaz de atender em uma fatia de tempo e gerenciando os processos hábeis de acessar o disco, é possível garantir uma reserva de banda de disco no nível do usuário.

PUC-Rio - Certificação Digital Nº 0521501/CA

3.2.3 Reserva de Largura de Banda de Rede

Apesar da provisão de reserva de largura de banda de rede não ser um dos objetivos iniciais do ReservationSuite, implementar essa funcionalidade em nosso conjunto de ferramentas passou a ser uma tarefa simples, visto que boa parte da infraestrutura de controle de acesso ao disco poderia ser reutilizada para o controle de acesso à rede. Nessa infraestrutura já haviam sido implementados, por exemplo, códigos para o compartilhamento de informações entre a aplicação e o servidor ReservationSuite, para o tratamento de concorrência entre diferentes linhas de execução de uma única aplicação e para a redefinição de funções da `libC`. Basicamente, era necessário apenas reimplementar as funções

de acesso à rede e, assim, decidimos introduzir o controle de uso desse recurso de comunicação em nossa suíte de ferramentas.

O controle de largura de banda de rede no ReservationSuite é feita utilizando-se interposição de chamadas a operações de envio de dados tais como *send* e *sendto*. Como chamadas *write* também podem enviar dados à rede, elas também foram interpostas, mas com o cuidado de se verificar se o descritor de arquivos da chamada é correspondente a um arquivo comum ou a um *socket*. Por limitação de tempo na fase de experimentação, optamos por não sobrescrever as operações de recebimento de dados.

Ao contrário do que acontece em uma mídia de disco rígido, na rede, a garantia de reserva de largura de banda depende fortemente de um suporte para provisão de QoS fim-a-fim, não podendo esse tratamento ser garantido apenas pelo servidor final. Por esse motivo, a solução de reserva de largura de banda de rede implementada pelo ReservationSuite é apenas paliativa no sentido que, sozinha, não garante qualidade de serviço na rede. No Linux, uma alternativa ao uso do ReservationSuite consiste na ferramenta `tc` a qual manipula o controle de tráfego na rede baseada em uma complexa estrutura de filas, classes e filtros de requisições de acesso a esse recurso [66].

Assim como acontece na reserva de largura de banda de disco, na reserva de largura de banda de rede também é necessário haver um controle de admissão de novas reservas baseado no limite de transferência de *bytes* entre a máquina servidora e a máquina cliente.

3.2.4 Implementação de Novas Políticas

Diferentes regras podem ser empregadas no momento da escolha de qual processo deve ser o próximo a executar em uma determinada máquina. Além disso, para satisfazer os requisitos de algumas aplicações, uma política de escalonamento pode ser mais indicada que outra. A possibilidade de escolher entre diferentes políticas, ou mesmo combiná-las, é uma característica muito importante em sistemas de uso compartilhado. No entanto, essa facilidade não é facilmente encontrada em ferramentas no nível do usuário e em sistemas operacionais com abstrações de reserva. Nesses sistemas, desenvolvedores de políticas de escalonamento precisam ter um conhecimento profundo sobre as estruturas de dados utilizadas pelo SO e sobre como os processos em execução são tratados. A prática da alteração de código no SO tende a gerar bons desempenhos, mas pode ser uma tarefa altamente complexa ou mesmo inviável de ser realizada. Por esse motivo, decidimos adotar uma separação total entre as políticas de escalonamento e os mecanismos de reserva presentes no

PUC-Rio - Certificação Digital Nº 0521501/CA

ReservationSuite. Em nosso conjunto de ferramentas, enquanto todo o código é escrito em C, a política de escalonamento pode ser implementada em C ou em Lua, uma linguagem de *script* rápida, leve e fácil de ser embutida em outras linguagens [67]. Como uma linguagem de *script*, o código Lua é interpretado e não precisa ser compilado; como uma linguagem a ser embutida, Lua funciona como uma biblioteca que pode ser ligada a outras aplicações de modo a incorporar as facilidades da linguagem.

A parte estática do ReservationSuite, escrita em C, faz uso constante de chamadas de baixo nível ao SO. Essas chamadas são responsáveis por tarefas que nada dizem respeito às regras de escalonamento, tais como configurar alarmes, alterar a prioridade de processos e criar seções críticas. Forçar o desenvolvedor de novas políticas de escalonamento a entender o funcionamento de todas essas tarefas seria desnecessário por atribuir conceitos externos ao que as políticas se propõem a fazer. Potencialmente, essa mistura de conceitos poderia gerar inconsistências no funcionamento da ferramenta como um todo. Por isso, as políticas de escalonamento do ReservationSuite podem ser estendidas sem a necessidade de manipular o código fonte do ReservationSuite. É necessário somente implementar a função `SchedPolicy`.

Políticas Lua devem implementar a função `SchedPolicy` recebendo dois parâmetros: uma tabela populada com todos os dados dos processos sendo gerenciados (todas as propriedades da tabela 3.2) e a hora atual em milissegundos. Esses parâmetros contêm informações suficientes para a implementação de algumas políticas de escalonamento. Para verificar a flexibilidade do escalonamento, três políticas foram implementadas:

- Alarme mais próximo: retorna os *pids* dos processos cujos alarmes expiram primeiro;
- EDF (Earliest Deadline First): retorna o *pid* do processo cujo período de execução expira primeiro, priorizando aqueles processos que ainda não executaram suas fatias de tempo para o período corrente;
- Aleatória: retorna o *pid* de um processo escolhido aleatoriamente a cada 10 ms. Processos que ainda não executaram a sua fatia de tempo para o período corrente são priorizados.

Para mostrar o quão simples a implementação de uma política de escalonamento no ReservationSuite pode ser, dois códigos são apresentados, um em Lua e outro em C, ambos implementando uma política de alarme mais próximo. A política Lua é apresentada a seguir:

```
1 function SchedPolicy (now, processTable)
2     local nextPidsToExecute = {}
```

PUC-Rio - Certificação Digital Nº 0521501/CA

```
3     local min = processTable[1].alarmTime
4
5     for i=2,#processTable do
6         if(processTable[i].alarmTime <= min) then
7             min = processTable[i].alarmTime
8         end
9     end
10
11    for i=1,#processTable do
12        if((processTable[i].alarmTime == min) or
13          (processTable[i].alarmTime <= now)) then
14            table.insert(nextPidsToExecute,
15                         processTable[i].pid)
16        end
17    end
18
19    return nextPidsToExecute
20end
```

Nessa política, primeiramente, na linha 2, uma tabela vazia (`nextPidsToExecute`) é criada para armazenar os *pids* dos próximos processos a serem tratados pela *thread* de alarme (a mesma descrita na seção 3.2.1). Na linha 5, a política itera sobre a lista de processos com a finalidade de achar o menor valor de `alarmTime` para, em seguida, na linha 11, popular a tabela `nextPidsToExecute` com os *pids* dos processos cujos alarmes expiram nesse menor valor. Nota-se que processos cujos alarmes já expiraram e, por algum motivo, não foram tratados, também são inclusos na tabela.

A seguir, a mesma política de alarme mais próximo é implementada em C:

```
1void SchedPolicy(GHashTable *processList,
2         unsigned int *pids, unsigned long now) {
3     GHashTableIter iter;
4     gpointer key, value;
5     struct processDataStruct *procNode;
6     unsigned long min;
7     int i;
8
9     int first = 0;
10    g_hash_table_iter_init(&iter, processList);
```

PUC-Rio - Certificação Digital Nº 0521501/CA

```
11    while (g_hash_table_iter_next(&iter, &key, &value)) {
12        procNode = (struct processDataStruct *) value;
13        if (first == 0) {
14            min = procNode->alarm_time;
15            first = 1;
16        } else {
17            if (procNode->alarm_time < min)
18                min = procNode->alarm_time;
19        }
20    }
21
22    i = 0;
23    g_hash_table_iter_init(&iter, processList);
24    while (g_hash_table_iter_next(&iter, &key, &value)) {
25        procNode = (struct processDataStruct *) value;
26        if((procNode->alarm_time == min) ||
27           (procNode->alarm_time <= now))
28                pids[i++] = procNode->pid;
29    }
30}
```

PUC-Rio - Certificação Digital Nº 0521501/CA

No política C, a função `SchedPolicy` recebe como parâmetro uma *hashtable* (implementada pela *Glib*) com a lista de processos ativos, uma lista que retornará os *pids* dos processos com os alarmes mais próximos e a hora atual em milissegundos. No laço iniciado na linha 12, a política procura pelo menor alarme entre os processos e os armazena na lista *pids* (linha 16 e 28).

Nota-se que não são necessários testes para verificação do tamanho da tabela de processos nos códigos das políticas. Essa tarefa é realizada por um procedimento padrão que deve estar contido em todos os arquivos que implementam a função `SchedPolicy`.

3.2.5
Combinando Múltiplas Políticas

Às vezes, é importante tratar as aplicações de um servidor de maneira diferenciada, aplicando regras específicas para cada tipo de aplicação ou usuário. Considere, por exemplo, o caso de um servidor de aplicação usado por duas classes de usuários, uma pagante e outra não-pagante. Tratar ambas as categorias de maneira equivalente pode ser justo, mas pode reduzir os ganhos do provedor de serviço. Uma solução para esse caso pode ser o uso de um escalonador hierárquico.

PUC-Rio - Certificação Digital Nº 0521501/CA

Um escalonador hierárquico pode, a cada decisão do escalonamento, utilizar uma política inicial para escolher qual classe de aplicações deve executar no próximo período. Uma vez que a classe a ser executada estiver sido definida, o escalonador aplica uma política de escalonamento para decidir qual aplicação da classe escolhida deve executar. A vantagem dessa prática hierárquica é que ela permite que cada classe de aplicação aplique suas próprias regras no uso dos recursos alocados a ela.

A fim de verificar a viabilidade de um escalonamento hierárquico tal como o descrito, prototipamos uma versão do ReservationSuite habilitada somente com o controle do recurso de processamento. Poucas alterações foram necessárias no servidor a fim de: i) permitir que o servidor carregue mais de uma política de escalonamento simultaneamente; ii) criar um controle de admissão para cada política carregada; iii) criar uma lista de processos para cada classe de aplicação.

A figura 3.6 apresenta um exemplo de uma hierarquia de políticas. Primeiramente, toda vez que um processo é criado ou a cada novo período (10 ms, por exemplo), um novo ciclo de escalonamento deve ser realizado. Nesses momentos, o servidor decide qual processo deve ser executado no próximo período. Antes de escolher um processo específico, o servidor invoca uma política para determinar qual classe de aplicações deve ser ativada. A política que implementamos é aleatória e atribui 80% de chances de execução aos clientes pagantes e 20% de chances aos demais clientes. Quando uma classe é escolhida, a decisão de escalonamento é passada para uma política local ao tipo de aplicação. No exemplo da figura 3.6, usamos a política Rate Monotonic (RM) para os clientes pagantes, pois essa política apresenta um tempo de resposta mínimo e constante. Para os clientes não-pagantes, utilizamos a política Earliest Deadline First (EDF), a qual não garante um tempo de resposta mínimo, mas provê um uso mais eficiente do processamento quando comparada à política RM [68].

3.3 Considerações Finais

Neste capítulo, através da implementação do conjunto de ferramentas de reserva ReservationSuite, mostramos que existem mecanismos providos por sistemas operacionais de propósito geral que auxiliam na garantia e na limitação do uso de recursos computacionais sem que alterações no núcleo do SO sejam necessárias. Esses mecanismos são bastante eficazes no controle de uso do processamento, mas apenas paliativos no controle da largura de banda de E/S utilizada. Isso porque não são capazes de anular as decisões de

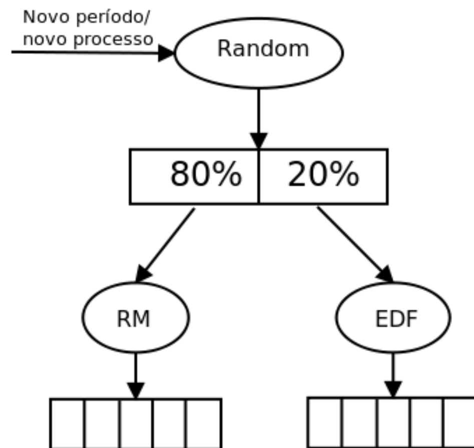

Figura 3.6: Uma hierarquia de políticas.

PUC-Rio - Certificação Digital Nº 0521501/CA

gerenciamento ditadas pelo SO. Assim, a provisão de novos escalonadores nos sistemas operacionais, com novas APIs para o controle de acesso aos recursos é essencial para a melhoria no gerenciamento de recursos no nível do usuário. Em especial, é preciso que a priorização do acesso ao disco seja integralmente cumprida sem que reordenações a essa mídia ocorram por conta de otimizações do SO. Iniciativas tais como o *framework* são importantes, mas ainda pouco configuráveis para serem utilizadas por aplicações com diferentes padrões de uso de recursos. Dessa maneira, a externalização do controle de uso de recursos pode ser uma tendência nos sistemas operacionais monolíticos à medida que sistemas operacionais de micronúcleo têm se mostrado bastante eficazes na tarefa de flexibilizar o gerenciamento de seus recursos.

Da forma como foi implementado, o ReservationSuite apresenta algumas restrições de uso advindas, principalmente, da necessidade do servidor ser executado com privilégios de administrador e do compartilhamento da árvore de diretórios do sistema entre os diferentes processos em execução, visto que todos os processos enxergam o mesmo diretório raiz do usuário *root*. Essas questões consistem em brechas de segurança e precisam ser contornadas.

Em sistemas de uso controlado, onde os usuários são previamente conhecidos, as questões citadas podem não ser fatores limitantes ao uso do nosso conjunto de ferramentas. Mesmo assim, nesses casos, pode-se atenuar o problema do compartilhamento de informações criando-se novos diretórios de execução com o comando *chroot*. Em ambientes onde o controle de usuários é menor e a segurança deve ser reforçada, é possível remover as partes da ferramenta que necessitam de privilégios administrativos. Essas partes, utilizadas para aumentar a prioridade de processos no acesso à CPU e ao disco, se removidas, não influenciam na limitação do uso de recursos, mas podem diminuir a eficácia da

garantia das reservas, pois todos os processos passam a ter chances parecidas de acesso aos recursos.

A decisão de implementar reservas de largura de banda de disco anteriormente à reserva de memória foi motivada pelo desafio de tratar uma mídia de acesso tão lento como o disco rígido. Devido à dificuldade envolvendo o controle de acesso ao disco, era necessário que a reserva dessa mídia fosse bem implementada e devidamente avaliada sem a interferência de gargalos relativos ao uso de memória, um recurso que, quando escasso, interfere diretamente na taxa de utilização do disco. Assim, optamos por não implementar ferramentas para a reserva de memória, mas é importante notar que sistemas operacionais de propósito geral já apresentam mecanismos de reserva desse recurso e que poderiam ser incorporados no ReservationSuite. No Linux, por exemplo, há o *framework* CGroup e o comando *setrlimit*. No entanto, desafios na reserva de memória ainda existem. Um exemplo consiste na dificuldade de, baseado no montante de memória utilizada por uma biblioteca compartilhada, determinar com quanto desse montante cada aplicação que faz uso da biblioteca deve arcar.

O fato do ReservationSuite ser implementado no nível do usuário impede o seu uso para gerenciar aplicações fortemente de tempo real (do tempo inglês *hard real-time*). Para esse domínio de aplicações, sistemas operacionais específicos devem ser utilizados. O uso do ReservationSuite é indicado para aplicações levemente de tempo real (do tempo inglês *soft real-time*) e aplicações que requerem isolamento de desempenho, perfis muito encontrados em cenários modernos de computação. Aplicações cujos códigos manipulam a sua própria prioridade de acesso aos recursos, redefinem funções da `libC` ou bloqueiam o recebimento de sinais também não devem ser gerenciadas pelo ReservationSuite.

Políticas de escalonamento, escritas nas linguagens C ou Lua, também podem ser facilmente implementadas e empregadas no servidor do ReservationSuite. Para exemplificar tal facilidade implementamos três políticas. Apesar de simples, essas políticas serviram como meios de compreender melhor o funcionamento de nossa ferramenta. No entanto, políticas mais complexas, tais como a baseada em loteria e a *Borrowed-Virtual-Time* [68, 69, 70], também podem ser implementadas no ReservationSuite sem maiores problemas.

Quando se fala em qualidade de serviço ao cliente é preciso considerar a qualidade de serviço provida pela rede e a qualidade provida pelo sistema do servidor. Neste trabalho, apenas o último tipo de qualidade é abordado, por termos como alvo um nicho de aplicações cujo desempenho é predominantemente influenciado pelos tempos de resposta do servidor. A união das qualidades de serviço na rede e nos servidores finais, chamada de provisão de

PUC-Rio - Certificação Digital Nº 0521501/CA

QoS fim-a-fim, necessita de uma integração maior de mecanismos de reserva.

Nos próximos capítulos, experimentos envolvendo a eficácia e o desempenho das técnicas de provisão de reservas são apresentados. O objetivo é mostrar que as reservas providas pelo ReservationSuite são eficientes em diferentes cenários, desde que algumas restrições de uso sejam devidamente respeitadas.

PUC-Rio - Certificação Digital Nº 0521501/CA