Controle Dinâmico de Recursos em Sistemas Operacionais

Márcio Starke, Carlos Maziero e Edgard Jamhour

PPGIA – Programa de Pós-Graduação em Informática Aplicada
PUCPR – Pontifícia Universidade Católica do Paraná
{marcio,maziero,jamhour}@ppgia.pucpr.br

***Resumo**: Uma função importante dos sistemas operacionais é a gerência dos recursos do computador usados pelas aplicações. Os mecanismos de gerência oferecidos pelos sistemas operacionais de mercado são limitados, permitindo apenas a definição de limites estáticos ou prioridades no uso dos recursos. Este artigo define mecanismos flexíveis para a gerência de processador, permitindo dimensionar os recursos disponíveis a um processo ou grupo de processos. O mecanismo é dinâmico e permite alterar a quantidade alocada de recursos durante a execução dos processos, em resposta a alguma necessidade específica. O protótipo atual permite a definição e ajuste dinâmico de percentuais mínimos e máximos de uso do processador por processos, grupos de processos, usuários ou grupos de usuários do sistema. Alguns resultados preliminares são apresentados.*

***Palavras-chave**: alocação de recursos, reserva de processador.*

***Abstract**: a relevant function of an operating system is to manage local resources used by applications. Management mechanisms provided by COTS operating systems generally offers only the definition of static limits or priorities in resource usage. This paper defines flexible mechanisms to manage the processor allocation, allowing defining the amount of processor available to a process or process group. The proposed mechanism allows redefining the processor allocation at any time, in response to specific needs. The current prototype permits setting upper and lower levels of processor usage for processes, process groups, users, and user groups. Some preliminary results are shown.*

***Keywords**: resource allocation, processor reservation.*

1. Introdução

Os computadores modernos contêm uma grande variedade de recursos como processadores, memórias, discos, interfaces de rede, impressoras, etc. Uma das principais responsabilidades dos sistemas operacionais é a de gerenciar o uso dos recursos do computador pelos processos em execução. Estes recursos compreendem entidades concretas, como espaço em disco, memória RAM e tempo de processamento, e também entidades abstratas, como conexões de rede, arquivos abertos, semáforos, etc. Em um sistema computacional com muitos usuários simultâneos, como um servidor de e-mail ou de arquivos, a necessidade de gerenciar o uso dos recursos compartilhados é ainda maior. Todavia, os mecanismos de gerência oferecidos pelos sistemas operacionais de mercado são muito limitados, geralmente permitindo apenas a definição de limites estáticos ou prioridades no uso desses recursos. Essa deficiência é muitas vezes usada como facilitador de ataques de negação de serviço em sistemas conectados à rede.

O objetivo principal do gerenciamento de recursos é prover oportunidade e garantia de acesso aos recursos, ao mesmo tempo em que os protege. Os recursos do sistema não são completamente independentes uns dos outros, assim, existem situações onde o acesso a certos recursos compromete novos acessos a ele mesmo ou a outros recursos [Glosh e Rajkumar 2002]. Uma situação onde a interdependência dos recursos pode ser notada é no tratamento dos pacotes de rede. Quando a interface de rede recebe um pacote, ela faz uma requisição para o sistema operacional executar as ações necessárias para transferência do pacote para o seu destino. Porém, quando o processador está saturado, essa requisição pode demorar a ser atendida, e o pacote pode ser descartado. Isso ocorre principalmente em interfaces de rede muito rápidas, como as interfaces *Gigabit ethernet*, em computadores com processadores lentos.

Dessa forma, o administrador do sistema precisa dispor de mecanismos para controlar a disponibilidade dos recursos aos processos, de acordo com a importância relativa de cada processo. Como essa importância pode ser variável ao longo do tempo, os mecanismos de controle devem permitir ajustar dinamicamente a quantidade de recursos disponível a cada processo [Nagar et al 2003]. Por exemplo, o sistema operacional deve permitir que se reserve um percentual adequado do uso do processador aos processos do administrador, mesmo em situações de elevada carga de processamento. Outro exemplo hipotético seria permitir reduzir ao mínimo o uso do processador por um processo que esteja sob ataque, visando permitir a observação e análise desse ataque. Tornando possível garantir quantidades mínimas de recursos a determinados processos, mecanismos para obtenção de Qualidade de Serviço (QoS) podem ser mais facilmente implementados [Bentaleb e Bétourné 1997].

Este artigo está organizado da seguinte forma: a seção 2 enumera os principais mecanismos de gerência de recursos dos sistemas atuais; a seção 3 define a política de gerência proposta; a seção 4 apresenta a implementação do protótipo atual, a seção 5 apresenta os resultados obtidos até o momento e a seção 6 apresenta e discute trabalhos correlatos.

2. Gerência de recursos

No âmbito do controle do uso de recursos por um processo ou usuário, é possível classificar os recursos do sistema em dois grupos:

- *Recursos preemptíveis*: são recursos que podem ser alocados a um processo e retirados dele dinamicamente, sem corromper a continuidade de sua execução. Um exemplo típico desse tipo de recurso é o processador, que é entregue a um processo para execução e retirado do mesmo em seguida, para ser entregue a outro processo[1]. Outros recursos que se encaixam nessa categoria são a quantidade de memória física (RAM) e a banda de rede disponíveis ao processo.
- *Recursos não-preemptíveis*: são recursos que, uma vez fornecidos a um processo, não podem ser retirados dele sem comprometer a continuidade de sua execução. Exemplos desse tipo de recurso são a quantidade de memória total (real + virtual) e o número de arquivos abertos disponíveis ao processo. Limitar o uso desses recursos a patamares muito baixos pode impedir a execução de processo, caso ele

[1] Essa classificação não se aplica diretamente a sistemas de tempo real, onde a disponibilidade de processador e memória física em quantidade suficiente pode ser crítica para a execução de um processo.

tenha negados seus pedidos de abertura de arquivos ou alocação de áreas de memória.

Outra característica que distingue esses dois grupos de recursos é a possibilidade de ajuste dinâmico em seu uso. A qualquer momento é possível aumentar ou diminuir a quantidade de tempo de processador disponível a um processo sem impedi-lo de executar (obviamente sua velocidade de execução irá variar de acordo com essa disponibilidade). No entanto, o mesmo ajuste não pode ser feito com relação ao número de arquivos abertos por um processo. Por exemplo, não é possível limitar em 5 o número de arquivos abertos por um processo que já tem 8 arquivos abertos, ou que necessite abrir 8 arquivos para executar corretamente.

Os sistemas operacionais de mercado oferecem alguns mecanismos para o controle da quantidade de recursos do sistema disponível a cada processo ou usuário, mas estes funcionam de uma forma relativamente primitiva [Mercer e Rajkumar 1995]. Por exemplo, o sistema operacional Linux oferece mecanismos para limitar a quantidade de memória utilizada por um processo, além de outros recursos. Através do sistema PAM (*Pluggable Authentication Modules* [Samar 1996]) é possível definir limites superiores para recursos como:

- `fsize`: tamanho máximo de arquivo
- `nofile`: número máximo de arquivos abertos
- `stack`: tamanho máximo da área de pilha em memória
- `data`: tamanho máximo da área de dados em memória
- `cpu`: tempo máximo de utilização da CPU
- `as`: tamanho máximo do espaço de endereçamento de um processo

Esses limites podem ser aplicados a processos, usuários ou grupos de usuários. No entanto, esses limites são fixados no início de cada sessão de usuário e não são ajustáveis durante a sessão. Caso eles sejam alterados, os novos valores serão válidos somente para a próxima sessão do usuário. Além disso, esse mecanismo somente permite a definição de limites máximos do uso de cada recurso. Não é possível garantir uma quantidade mínima de recursos a um processo ou grupo de processos independente da carga do sistema.

3. Modelo de Controle de Recursos Proposto

A proposta deste projeto é definir e implementar um mecanismo de controle dinâmico de recursos preemptíveis para um sistema operacional multi-usuário. Por controle entende-se a possibilidade de definir limites inferiores e superiores na disponibilidade de recursos aos processos existentes no sistema. Esses limites devem poder ser estabelecidos e reajustados dinamicamente, a qualquer momento da vida dos processos alvo, não necessitando suspendê-los ou reiniciá-los para a aplicação dos limites.

Os diagramas da figura 1 exemplificam os conceitos de limite inferior e superior no uso dos recursos. No diagrama à esquerda, três processos (P1, P2 e P3) com as mesmas características e mesma prioridade disputam o tempo de processador, ficando cada um com 33% do mesmo. Ao ajustar o limite inferior de P1 para 60% em *t=2*, este passa a usar 60% do tempo de processador, enquanto os outros 40% do recurso são divididos entre P2 e P3. No caso do diagrama à direita ocorre o inverso: o limite superior de tempo de

processamento de P1 é ajustado para 10%; neste caso, P2 e P3 dividem entre si o percentual restante (90%).

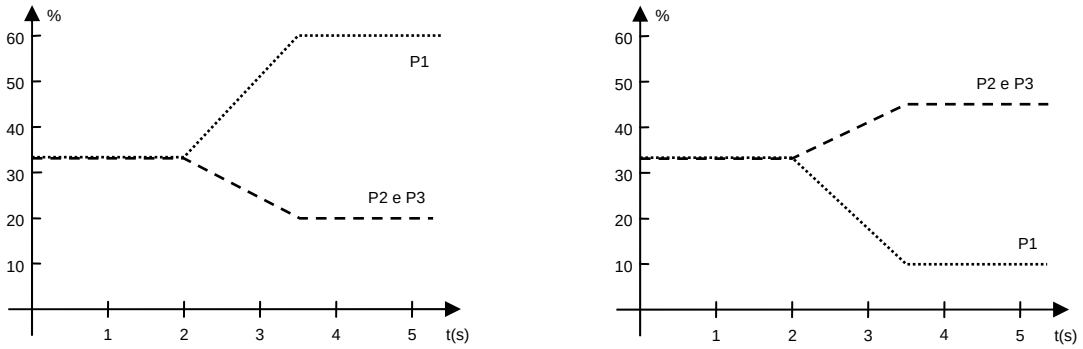

Figura 1. Aplicação de limite inferior (esquerda) e superior (direita) no uso do processador.

Nos diagramas da figura 1 observa-se que a mudança na distribuição do recurso entre os processos não ocorre imediatamente. Devido à natureza dos algoritmos de escalonamento dos recursos em jogo (processador, memória física e outros recursos preemptíveis), existe um período de transição durante o qual o sistema deve se adequar às novas restrições. Um dos objetivos do mecanismo proposto é buscar formas de minimizar a duração desse período de transição.

Normalmente o controle do uso dos recursos é feito com base em processos, visto que estes são a unidade básica de alocação de recursos nos sistemas operacionais correntes[2]. Algumas vezes, no entanto, pode ser necessário definir regras de alocação de recursos para grupos de processos relacionados (por exemplo, os processos de uma mesma aplicação). Assim, a definição de regras de alocação de recursos deve considerar diferentes níveis de agrupamento de processos. Os níveis considerados na presente proposta são apresentados na figura 2.

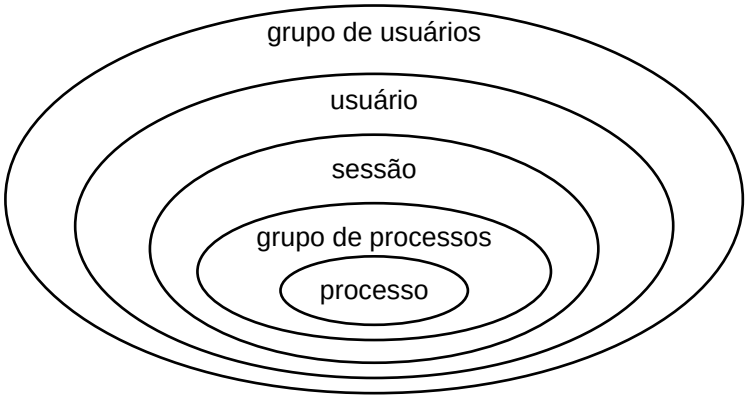

Figura 2. Granularidade do sistema de controle dinâmico de recursos

[2] Isso pode variar dependendo da natureza do recurso em questão. Por exemplo, enquanto a unidade básica de alocação de memória é o processo, em núcleos que implementem modelos de *threading* 1-1 ou M-N a unidade básica de alocação do processador é a *thread*, e não o processo.

O *grupo de processos* indicado na figura 2 é uma coleção de processos que possuem um identificador de grupo (*group id*) em comum. Esse identificador é fornecido pelo processo chamado "líder do grupo". Qualquer processo pode tornar-se líder, bastando executar a chamada de sistema apropriada. Tal ação gerará um novo identificador de grupo e o tornará líder de seu próprio grupo [GNU C Library Manual 2004].

Analogamente, uma *sessão de processos* define um conjunto de grupos de processos que compartilham um mesmo identificador de sessão (*session id*), pertencente originalmente ao líder da sessão. Para um processo tornar-se líder de sessão basta executar a chamada de sistema apropriada, gerando um novo identificador de sessão.

A política de acesso às regras de definição de limites é feita com base no usuário que ativa a regra. Um usuário pode restringir o acesso aos recursos de seus próprios processos, ou seja, ele pode definir regras aplicáveis sobre *threads*, processos, sessões ou grupos de processos de sua propriedade, contanto que sejam regras estabelecendo limites superiores no uso dos recursos. Somente o administrador do sistema poderá criar ou alterar regras definindo limites inferiores, ou regras sobre usuários ou grupos de usuários.

No momento da definição de regras de limite inferior, deve-se verificar que a soma dos limites de todas as regras de limite inferior não ultrapasse a quantidade disponível do recurso. Por exemplo, não é possível criar regras definindo limites inferiores de uso de processador nas quais a soma dos limites seja superior a 100%. Portanto, regras que excederem a capacidade disponível do recurso em questão devem ser descartadas.

Deve-se ter em mente que, ao definir regras de limite inferior, somente a reserva de recursos é feita. A sua utilização efetiva dependerá do processo (ou grupo) ao qual a regra se aplica, pois o núcleo do sistema somente pode garantir que o recurso estará disponível caso seja necessário, e não pode forçar o processo a efetivamente utilizar aquele recurso caso não o necessite.

4. Implementação da proposta

Para verificar a validade da proposta foi implementado um protótipo capaz de controlar dinamicamente a alocação do tempo de processador entre vários processos ou agrupamentos, conforme definidos na figura 2. A implementação foi efetuada através da modificação do núcleo do sistema Linux versão 2.4.21.

Para a definição de regras de alocação de recursos foi criada uma nova chamada de sistema denominada `resourcelimits(`*resource, scope, target, value, type*`)`. Os argumentos dessa nova chamada de sistema têm o seguinte significado:

- *resource*: recurso afetado pela regra (somente `CPUPERC` no protótipo atual)
- *scope*: escopo da regra, podendo assumir os seguintes valores: `RLPROCESS`, `RLPROCESSGROUP`, `RLSESSION`, `RLUSER` ou `RLGROUP`.
- *target*: identificador do processo, grupo de processos, sessão, etc.
- *value*: valor numérico do limite a ser imposto.
- *type*: tipo de limite a ser imposto, pode assumir os valores `UPPER` ou `LOWER`.

Caso a regra possa ser aplicada com sucesso, a chamada retorna 0 (zero). Caso contrário, um código de erro negativo é retornado. Entre os erros possíveis estão a inexistência do alvo (*target*) e a impossibilidade de definir um limite inferior, caso a soma

dos limites já definidos com o novo limite ultrapasse 100% de uso do recurso. Novas regras podem ser definidas a qualquer momento através da nova chamada de sistema. Para cada nova regra é criada uma estrutura de dados interna ao núcleo contendo a definição da regra e o grupo de processos ao qual ela se aplica.

As alterações necessárias no núcleo do sistema operacional concentram-se no escalonador de processos. O escalonador de processos do Linux é baseado em tempo compartilhado, isto é, cada processo recebe uma pequena fatia de tempo de execução. O Linux utiliza um escalonador com prioridade, no qual cada processo recebe uma prioridade estática que pode ser alterada no através do comando `renice` ou da chamada de sistema `nice`.

No Linux, um *tick* é definido como sendo o intervalo de tempo entre duas interrupções consecutivas do relógio do hardware. Um contador denominado `jiffies` indica quantos *ticks* já ocorreram desde a inicialização do sistema. A fatia de tempo (*quantum*) que cada processo recebe é um valor que corresponde à quantidade de *ticks* que o processo possui para sua execução. A cada *tick* decorrido o processo em execução tem o seu *quantum* decrementado; quando este atinge zero, o escalonador recalcula a prioridade efetiva do processo, provê uma nova fatia de tempo e coloca o processo novamente na fila de processos prontos.

Ao criar uma nova regra, é armazenado o valor atual de `jiffies` e calculado o instante `RLNEXT` em que as informações de escalonamento dos processos de seu grupo deverão ser atualizadas, pela seguinte forma: `RLNEXT = jiffies + HZ` (`HZ` é uma constante com valor 100 no kernel 2.4). Esse intervalo de tempo é denominado *iteração*. A cada troca de contexto, a quantidade de *ticks* de processamento utilizada pelo processo em execução é contabilizada para verificar se seu grupo já alcançou seu limite máximo de processamento dentro daquela iteração. Esse limite é calculado dinamicamente, levando em conta a regra de limite superior de seu grupo e as regras de limite inferior dos demais grupos. Caso o limite tenha sido alcançado, os processos daquele grupo não são mais escalonados até o final da iteração corrente.

Também foi necessário alterar a função `do_fork()`, que implementa a criação de processos, para registrar os novos processos nos grupos de processos afetados pelas regras existentes, caso eles sejam filhos de processos afetados pela regras.

5. Avaliação do protótipo

O protótipo foi testado em um servidor Athlon XP 1700+, com 256 MBytes de memória, rodando a distribuição Slackware 9 com o kernel 2.4.21. Foi criado um pequeno programa com um laço infinito, chamado `usecpu`, para que se pudesse avaliar eficácia do sistema. As informações sobre uso do processador foram obtidas pelo utilitário `top`, com taxa de atualização de 1 segundo.

Um primeiro experimento consistiu em executar vários processos `usecpu`, sendo aplicado um limite superior de uso da CPU a um deles. A figura 3 apresenta o comportamento temporal de um processo P_1 limitado a no máximo 10% de uso do processador, disputando-a com outros processos. São apresentadas curvas para N processos em execução, com N variando de 1 (somente P_1) a 6 (P_1 e mais 5 processos). A regra de definição do limite é aplicada sobre P_1 em *t=0*.

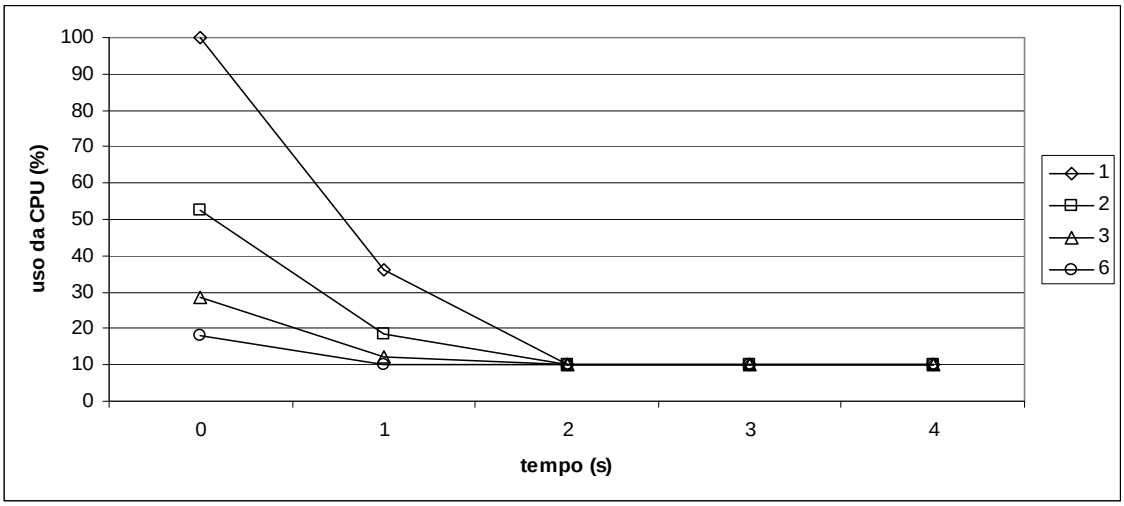

Figura 3. Uso da CPU por P_1, com N processos disputando a CPU.

Pode-se observar que o sistema leva cerca de 2 segundos para estabilizar o processo no limite estabelecido. Esse período de transição decorre do algoritmo de escalonamento do Linux e da própria granularidade das medidas efetuadas (1 segundo).

O mesmo experimento foi repetido com grupos de processos, sessões e usuários, com resultados bastante similares. Em nenhuma das execuções realizadas algum processo com limite máximo imposto excedeu o especificado pela regra.

Para os testes com limites mínimos de utilização do processador foi empregado o mesmo processo `usecpu`. Fixando o limite inferior de uso da CPU para P_1 em 90% e variando o número total de processos disputando o processador (de 1 a 6), foram obtidas as curvas apresentadas na figura 4. Nela pode-se observar uma dinâmica de adaptação do sistema similar à do experimento anterior.

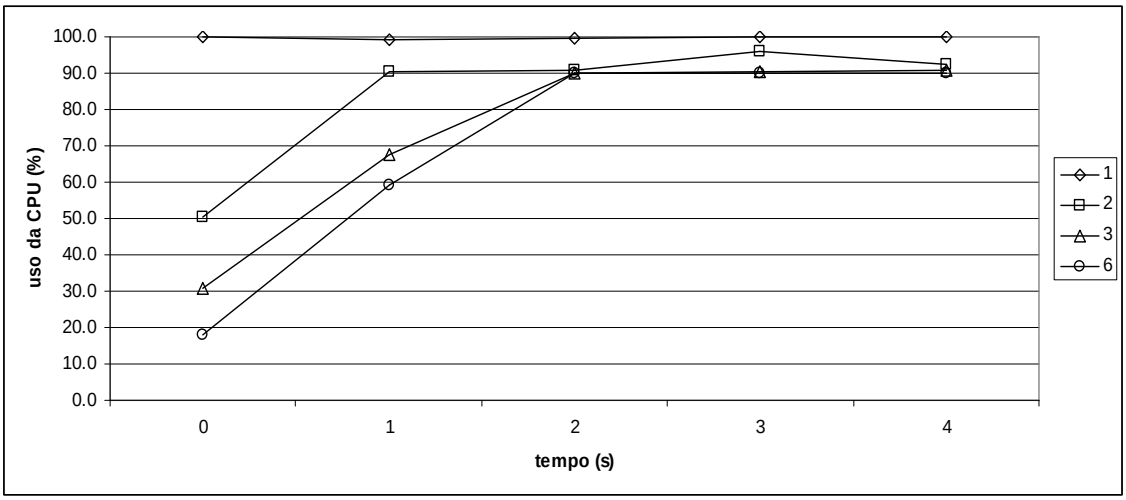

Figura 4. Uso da CPU por P_1, com N processos disputando a CPU.

A sobrecarga de processamento no kernel do Linux imposta pela interpretação das regras também foi avaliada. Para tal utilizada a ferramenta *lmbench* de análise de desempenho em ambientes Unix [McVoy e Staelin 1996]. Os parâmetros analisados foram o número de processos disputando a CPU (de 2 a 256) e tamanho dos processos em memória (de 128 Kbytes a 512 Kbytes). O gráfico da figura 5 apresenta os resultados

obtidos. Nele pode-se observar que o custo representado pela aplicação das regras permanece relativamente constante com o aumento da carga do sistema, permanecendo normalmente abaixo de 1% do tempo total de processamento.

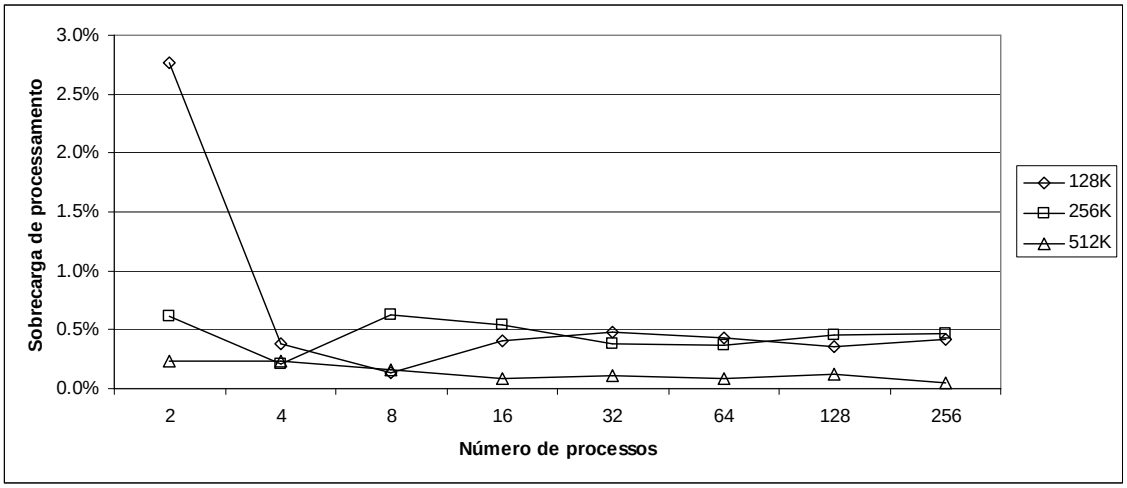

Figura 5. Acréscimo de tempo de execução
imposto pelo mecanismo de aplicação das regras.

Como pode ser visto, em nenhum momento o algoritmo deixou de cumprir os limites definidos nas regras. Também é possível comprovar que o modelo implementa um mecanismo dinâmico, visto que as alterações nas regras são aplicadas quase que instantaneamente, sem que seja necessário reiniciar os processos.

6. Trabalhos correlatos

Nesta seção serão apresentados e discutidos alguns trabalhos disponíveis na literatura ou na Internet que visam atingir os mesmos objetivos da proposta aqui apresentada.

O projeto CKRM – *Class-Based Kernel Resource Management* [Nagar et al 2003] busca desenvolver no kernel do Linux mecanismos para prover serviços diferenciados para os recursos como processador, páginas de memória, entrada/saída e banda de rede. Os processos são agrupados em classes, às quais podem ser atribuídos níveis de serviço diferenciados. O administrador do sistema pode criar classes, atribuir processos e usuários às classes e definir níveis de uso dos recursos às mesmas.

O sistema CKRM somente permite definir limites superiores, não permitindo a definição de limites inferiores no uso dos recursos. Além disso, o CKRM optou por criar sua própria sistemática para agrupar processos (através de classes), ao invés de utilizar os mecanismos presentes no sistemas Unix, como sessões e grupos de processos.

O objetivo do *Linux Soft Real-Time* [Childs e Ingram 2001] é prever a alocação de recursos a processos, sem no entanto garantir que a alocação requerida vai ser obtida. No Linux padrão o mecanismo de escalonamento de processador é feito através do melhor esforço. O Linux-SRT define uma nova política de escalonamento que permite às aplicações especificarem uma quantidade fixa de uso do processador em um determinado período. Foram implementados mecanismos para oferecer uma política similar para controlar o uso da banda de acesso a disco. Esse trabalho é voltado para aplicações que

necessitam de garantias de QoS, portanto o controle dos recursos é realizado sobre aplicações específicas, não permitindo operar sobre grupos de processos ou usuários.

O sistema *CPU – Cap Processor Usage* [Golab 2004] consiste de um *patch* para o kernel do Linux que permite limitar o quanto um processo pode utilizar o processador. Esse *patch* modifica a tabela de controle de processos do Linux, adicionando a ela informações sobre os limites de uso do processador para aquele processo. Ele define como unidade de alocação dos recursos somente processos, sem possibilidade de agrupamentos. Também não é possível definir limites inferiores para os processos.

O sistema *DSRT – Dynamic Soft Real-Time CPU Scheduler* [Jackson 2002] implementa um escalonador de processador em nível de usuário (ou seja, for a do núcleo) capaz de garantir restrições de processamento em aplicações de tempo real. Esse sistema oferece a possibilidade de definir limites inferiores (mínimos) de tempo de CPU com base em várias estratégias, fixas ou adaptativas. Não há suporte para a definição de limites superiores (máximos) de uso do processador.

Os autores deste artigo não têm conhecimento de outros sistemas que permitam a definição conjunta de limites inferiores e superiores na alocação de recursos aos processos de um ambiente computacional.

7. Conclusões e Trabalhos Futuros

Neste artigo foi apresentado um modelo para controle dinâmico de recursos do sistema operacional. Esse modelo permite ajustar a quantidade de recursos preemptíveis alocadas aos processos durante sua execução. O modelo define limites inferiores e superiores para o uso dos recursos, e também permite agregar os processos de forma a aplicar limites sobre grupos de processos relacionados.

O protótipo implementado, que aplica o modelo proposto ao controle do uso do processador em um sistema Linux, permitiu verificar a viabilidade da proposta. Em todos os testes realizados o comportamento dos processos correspondeu ao esperado. Além disso, o impacto do mecanismo sobre o desempenho do sistema é modesto.

O modelo aqui proposto tem várias possibilidades de uso. Com ele, aplicativos com necessidades de QoS podem ser mais facilmente implementados, visto que o modelo já considera as primitivas necessárias à implementação. Ele também permite implementar mecanismos de respostas a ataques, limitando a execução de processos que estejam comprometidos e com isso preservando o valor forense das informações do sistema [Monteiro 2003].

Como trabalhos futuros considera-se a extensão do protótipo para implementar o controle de outros recursos preemptíveis, como quantidade de memória física (RAM), banda de disco e de rede. Também devem ser realizadas otimizações no código para diminuir o custo do mecanismo e também o período de transição após cada modificação da alocação de recursos.

Referências

Bentaleb, H., Bétourné, C. *QoS generic mechanisms in an operating system*. WFCS'97 IEEE International Workshop on Factory Communication Systems, 1997.

Childs, S., Ingram, D. *The Linux-SRT integrated multimedia operating system: bringing QoS to the desktop*. RTAS'01 IEEE Real-Time Technology and Applications Symposium, 2001.

Nagar, S., Franke, H., Choi, J., Seetharaman, C., Kaplan, S., Singhvi, N., Kashyap, V., Kravetz, M. *CKRM – Class-based prioritized resource control in Linux*. Ottawa Linux Symposium, 2003.

Glosh, S., Rajkumar, R. *Resource management of the OS network subsystem*. 5th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, 2002.

GNU C Library Manual. *Process group functions*. Na internet: http://www.gnu.org/software/libc/manual, 2004.

Golab, K. *CPU – Cap Processor Usage*. TLS Technologies. Na internet: http://www.tls-technologies.com/CPU/cpu-main.html, 2004.

Jackson, J. W. *Multiprocessor Soft Real Time CPU Resource Manager and its Validation with Hypermedia Applications*. Master's Thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, May 2002.

McVoy, L., Staelin, C. *Lmbench: Portable tools for performance analysis*. Winter USENIX Conference, 1996.

Mercer, C., Rajkumar, R. *An interactive interface and RT-Mach support for monitoring and controlling resource management*. RTAS'01 IEEE Real-Time Technology and Applications Symposium, 1995.

Monteiro, D. *Resposta automática em um sistema de segurança imunológico computacional*. Dissertação de Mestrado, Instituto de Computação, Universidade Estadual de Campinas, 2003.

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

Samar, V. *Unified login with pluggable authentication modules (PAM)*. 3rd ACM Conference on Computer and Communications Security, 1996.

Tanenbaum, A., Woodhull, A. *Sistemas Operacionais: Projeto e Implementação*. Editora Bookman, 1997.