MARTIN ALAIN KRETSCHEK

PANALYSER, UMA FERRAMENTA DE BAIXO IMPACTO PARA MEDIÇÃO DE UTILIZAÇÃO DE RECURSOS DO SISTEMA OPERACIONAL LINUX

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre. Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.

Orientador: Prof. Dr. Roberto André Hexsel

CURITIBA
2002

UFPR
Ministério da Educação
Universidade Federal do Paraná
Mestrado em Informática

PARECER

Nós, abaixo assinados, membros da Banca Examinadora da defesa de Dissertação de Mestrado em Informática do aluno *Martin Alain Kretschek*, avaliamos o trabalho intitulado, *"Panalyser: Uma Ferramenta de Baixo Impacto para Medições de Utilização de Recursos do Sistema Operacional Linux"*, cuja defesa foi realizada no dia 15 de março de 2002, às quatorze horas e trinta minutos, no anfiteatro B do Setor de Ciências Exatas da Universidade Federal do Paraná. Após a avaliação, decidimos pela aprovação do candidato.

Curitiba, 15 de março de 2002.

Prof. Dr. Roberto André Hexsel
DINF/UFPR - Orientador

Prof. Dr. Rogério Drummond
UNICAMP

Profª. Dra. Cristina Duarte Murta
DINF/UFPR

AGRADECIMENTOS

Aos meus pais e meu irmão pelo seu apoio e carinho nos momentos mais decisivos de minha vida.

À minha esposa e filha por seu carinho e sua compreensão de que é preciso muita dedicação e tempo para se fazer algo bem feito.

Aos avós de minha filha por sua dedicação e carinho com a neta, que me permitiram trabalhar com tranquilidade.

Ao meu orientador prof. Roberto André Hexsel por ter acreditado em minha capacidade e me ensinado a ser um verdadeiro pesquisador.

À prof. Cristina Duarte Murta por me ensinar a apresentar os resultados de meus experimentos e por ter viabilizado os mesmos na forma de equipamentos e ferramentas.

Ao prof. Marcos Alexandre Castilho por seus ensinamentos sobre o Linux e por ceder equipamentos para a realização de meus experimentos.

A todos os professores do Departamento de Informática que apoiaram e confiaram em minha capacidade.

A meu pai e meu irmão pela revisão do primeiro rascunho desta dissertação.

Aos amigos Aldri Luiz dos Santos e Luis Carlos Erpen De Bona por seus inestimáveis conselhos e ajuda na elaboração desta dissertação.

Ao amigo Luciano Nascimento por ter me encorajado a investir nesta empreitada.

Aos amigos Pedro Sérgio e Estefânia por me lembrarem que em todo trabalho é necessário uma pausa.

A Sra. Marilena Tavares Casali por ter bondosamente cedido os meios para acomodar confortavelmente minha família enquanto realizava este trabalho.

A todos que acreditaram em mim o meu muito obrigado.

SUMÁRIO

LISTA DE FIGURAS iv

LISTA DE TABELAS v

RESUMO vi

ABSTRACT vii

1 INTRODUÇÃO 1
1.1 Motivação 1
1.2 Trabalhos Relacionados 3
1.3 Organização da Dissertação 8

2 AVALIAÇÃO DE DESEMPENHO 9
2.1 Técnicas de Avaliação de Desempenho 10
2.2 Métricas de Desempenho 11
2.3 Desempenho em Sistemas Operacionais 14
2.4 Ferramentas de Avaliação de Desempenho 17
2.4.1 Strace 17
2.4.2 Ferramentas Baseadas no /proc 18
2.4.3 WebMonitor 19

3 A FERRAMENTA PANALYSER 21
3.1 Informações Fornecidas pelo Panalyser 21
3.2 Parâmetros do Panalyser 23
3.3 Considerações Sobre a Implementação 24
3.4 Comparação com Outros Monitores 30
3.4.1 Comparação entre Panalyser e Atsar 32
3.4.2 Comparação entre o Panalyser e Strace 33
3.4.3 Vantagens do Panalyser 34
3.5 Emprego das Medidas de Utilização de Recursos do Sistema Operacional . 34

4 ESTUDO DE CASO 37
4.1 O Gerador de Carga SURGE 37
4.2 Ambiente Experimental 38
4.3 Resultados dos Experimentos 40
4.3.1 Vazão Máxima e Saturação 40

4.3.2 Vazão x Latência 41
4.3.3 Utilização do Processador 43
4.3.4 Page Faults 44
4.3.5 Chamadas de Sistema 45
4.3.6 Conclusão Sobre a Saturação do Apache 48

5 CONCLUSÃO E TRABALHOS FUTUROS 49

APÊNDICES 51

A PROGRAMA PANALYSER 52

REFERÊNCIAS BIBLIOGRÁFICAS 111

LISTA DE FIGURAS

2.1 Capacidade de um sistema. 13
2.2 Níveis de divisão do subsistema de E/S do *kernel*. 15
2.3 Cópia física em um *pipeline* de E/S simples no UNIX. 17

3.1 A estrutura de dados `rusage`. As setas indicam os campos utilizados na versão 2.4.16 do *kernel* do Linux. 22
3.2 Parâmetros de configuração. 24
3.3 Algoritmo de funcionamento do `panalyser`. 26

4.1 Vazão dos processos Apache no servidor e latência média para a finalização das requisições nos clientes. 40
4.2 Vazão dos processos Apache no servidor medida pelo `panalyser` e latência média para a finalização das requisições nos clientes. 42
4.3 Média de utilização de CPU para 840 até 2280 EU, nos domínios do usuário e sistema. 44
4.4 Número de *page faults*/s para 840 e 1560 EU (topo), e 1800 até 2280 EU (base). 45
4.5 Número de *reads*/s para 840 e 1560 EU (esquerda), e 1800 até 2280 EU (direita). 46
4.6 Número de *writes*/s para 840 e 1560 EU (esquerda), e 1800 até 2280 EU (direita). 46
4.7 Número de *forks*/s para 840 e 1560 EU (esquerda), e 1800 até 2280 EU (direita). 47
4.8 Número de *socketcalls*/s para 840 e 1560 EU (esquerda), e 1800 até 2280 EU (direita). 47

LISTA DE TABELAS

3.1 Tempo de execução nos domínios do usuário e sistema do programa de teste não rastreamento e sendo rastreado pelo `atsar`, `panalyser` e `strace`. . . . 30
3.2 Comparação entre a utilização de CPU nos domínios de usuário e sistema medidos pelo `panalyser` e `atsar`. 33
3.3 Número de chamadas de sistema efetuadas pelo programa de teste contabilizadas e classificadas pelo `panalyser` e `strace`. 33

4.1 Parâmetros de configuração do Apache. 39
4.2 Variação percentual gerada pelo `panalyser` na vazão e latência médias do Apache. 42
4.3 Vazão e latência do Apache medido pelo `panalyser` em 5 repetições do experimento. 43
4.4 Médias das medições dos recursos utilizados pelo Apache. 44

RESUMO

O Sistema Operacional (SO) torna disponíveis e controla os recursos de *hardware* de um sistema computacional para os processos. Os dados de utilização destes recursos são de grande relevância para (i) o desenvolvimento e avaliação de desempenho de aplicações; (ii) o diagnóstico de pontos de contenção; (iii) a caracterização de carga; e (iv) o levantamento de parâmetros para validação de modelos de um sistema. Este trabalho apresenta uma ferramenta chamada `panalyser`, para medição de utilização de recursos pelos processos no Sistema Operacional GNU/Linux. O `panalyser` é um monitor que opera em batelada, controlado por eventos e amostragem. A ferramenta fornece dados sobre a utilização de CPU, memória primária e secundária, e classificação e totalização das chamadas de sistema dos processos monitorados. Os processos monitorados podem ser quaisquer processos, exceto o processo `init()`. O monitoramento de um certo processo permite também a observação de todos os seus descendentes. O `panalyser` causa baixo impacto no funcionamento e desempenho do SO porque se baseia nas chamadas de sistema `ptrace`, `wait4` e `getrusage` do Linux, enquanto ferramentas como `atsar`, `ps` e `top` utilizam-se da leitura do pseudo sistema de arquivos `/proc` que requer mais recursos do SO. Como conseqüência da pouca interferência no funcionamento do sistema, as medições efetuadas com o `panalyser` apresentam baixa distorção. Além disso, o `panalyser` é portável para todas as plataformas de *hardware* suportadas pelo Linux. Um estudo de caso apresenta o uso do `panalyser` no monitoramento dos recursos do SO utilizados por um servidor Web Apache quando este é submetido a diferentes intensidades de cargas, desde cargas leves até a saturação.

ABSTRACT

The Operating System controls and coordinates the access to the resources of a computer system by the processes. Information regarding resource usage is of great relevance for (i) application development and performance evaluation; (ii) diagnosis of system bottlenecks; (iii) workload characterization; and (iv) the choice of model parameters and validation of system models. This dissertation presents a performance evaluation tool named `panalyser`, devised for the measurement of resource usage by processes in the GNU/Linux Operating System. `Panalyser` runs in batch mode and is an event and sample driven monitor. `Panalyzer` allows for the measurement of CPU utilization, primary and secondary memory utilization, and the accounting and classification of system calls executed by monitored processes. Any processes in the system, except `init()`, can be monitored, including all or some of the descendants of a given process. `Panalyser` causes a small overhead on system performance because it is based on `ptrace`, `wait4` and `getrusage` Linux system calls, whereas tools such as `atsar`, `ps` and `top` read the pseudo-file system `/proc`, which is a more expensive operation. As a consequence of the small intrusion, measurements made with `panalyser` minimize distortion in the results. A case study is presented where an Apache web server is monitored while being driven at differing workload levels, from light load to saturation.

CAPÍTULO 1

INTRODUÇÃO

Este trabalho apresenta uma ferramenta de baixo impacto para medição de utilização de recursos do Sistema Operacional (SO) pelos processos. O papel de um SO é mediar e multiplexar o acesso de múltiplos processos aos recursos providos pelo *hardware* [2, 24]. Um sistema computacional é composto por aplicações no espaço do usuário, SO, *hardware* dos computadores, topologia da rede e *hardware* de interligação em rede dos computadores.

Ferramentas para medição de utilização de recursos do SO são usadas para fornecer dados sobre aspectos específicos do desempenho de um sistema de computadores. Estes dados, por sua vez, necessitam ser relacionados e comparados, dentro de determinados critérios, para obter uma visão ao mesmo tempo geral e precisa do desempenho do sistema avaliado.

Os dados de utilização de recursos do SO em um sistema são de grande relevância para: projeto e desenvolvimento de aplicações e sua avaliação de desempenho; diagnóstico de pontos de contenção; ajuste e melhoramento do desempenho; caracterização de uma carga de trabalho; e levantamento de parâmetros, validação e dados para modelos. Estes aspectos são de suma importância para o fornecimento de critérios chaves para o projeto, a aquisição e o uso de sistemas de computadores [12].

1.1 Motivação

As ferramentas para medição de utilização de recursos do SO que são distribuídas correntemente têm a capacidade de medir a utilização de uma vasta gama de recursos, incluindo tempo de CPU, a memória (RAM e os discos), e os dispositivos da rede. No entanto, estas ferramentas normalmente fornecem dados sobre o sistema como um todo, sem discriminar os dados de utilização de recursos para cada programa em execução. Outro problema destas ferramentas é seu alto custo de execução, que pode interferir significativamente nas

medições efetuadas. Além disto o relacionamento e comparação dos dados fornecidos por elas podem não ser triviais. Existem também ferramentas desenvolvidas para avaliação de programas específicos, como por exemplo o `WebMonitor` [7]. Estas ferramentas, apesar de apresentarem boa precisão nas medidas efetuadas, são de difícil portabilidade por exigirem modificações nos programas a serem medidos e até mesmo no SO.

Com a finalidade de resolver os problemas mencionados, foi desenvolvida a ferramenta `panalyser` para medição da utilização de recursos pelos processos no ambiente do Sistema Operacional GNU/Linux. O `panalyser` fornece dados sobre a utilização de CPU, memória primária e secundária, e classificação e totalização das chamadas de sistema para um dado processo. O `panalyser` apresenta pouca sobrecarga no sistema, minimizando distorções no resultado das medições. As informações fornecidas permitem uma melhor comparação dos dados medidos devido a técnica de amostragem de dados empregada. Outro aspecto importante do `panalyser` é a possibilidade da medição de qualquer processo em execução, inclusive processos do SO.

Na implementação do `panalyser` foram consideradas a implementação e a demanda de recursos do SO das ferramentas distribuídas com os SOs atuais. De modo a não repetir as mesmas implementações e modos de operação daquelas, apresentando assim os mesmos custos de execução e as conseqüentes distorções nas medidas produzidas, foram usadas chamadas de sistema padrão do SO Linux. Isto proporciona portabilidade à ferramenta, além de não ser necessário modificar o SO. A implementação do `panalyser` também não necessita que sejam feitas modificações nos programas a serem medidos. A medição de processos e de seus processos filhos é outro ponto de destaque do `panalyser`, que permite a clara discriminação da utilização dos recursos do SO entre vários processos.

Este trabalho apresenta um exemplo do uso do `panalyser` para a medição do processo servidor Web Apache [20]. As medições foram efetuadas variando-se a intensidade da carga de requisições para o Apache, até atingir a saturação em um sistema bem balanceado para a carga usada [10]. O objetivo do experimento foi demonstrar a capacidade do `panalyser` em fornecer dados de utilização de recursos do SO pelo Apache, e assim mostrar porque o mesmo atingiu um estado saturado.

1.2 Trabalhos Relacionados

Os trabalhos relacionados a seguir apresentam estudos sobre a influência do SO no desempenho de sistemas de computadores, técnicas de medições, de avaliação e de otimização do desempenho de processos, sistemas de memória, protocolos TCP/IP e adaptadores de interface de rede. Neste trabalho, esses estudos são usados como referência para a avaliar e discutir cada um dos aspectos da ferramenta desenvolvida para medição de utilização dos recursos do SO, desde os tipos de dados coletados, a implementação da coleta de dados, até sua aplicação na medição de processos executados sobre o Linux.

Um monitor é uma ferramenta usada para observar a atividade de um sistema. Em geral, monitores observam o desempenho dos sistemas, coletam estatísticas sobre o desempenho, analisam os dados e mostram resultados. Os monitores podem ser implementados em *hardware, firmware* e *software* [12]. Os monitores implementados em *software* geralmente observam o desempenho dos sistemas através dos dados de utilização de recursos do Sistema Operacional, uma vez que o papel do SO é mediar e multiplexar o acesso de múltiplos processos aos recursos providos pelo *hardware.* Logo, o SO é um dos pontos mais sensíveis no desempenho de um sistema.

Pasquale et al., em [19], mostram que a maioria dos SOs não suporta adequadamente aplicações que geram tráfego intenso de entrada e saída (E/S), por não conseguirem transferir os dados eficientemente entre processos e o SO. Um novo algoritmo para a transferência de dados entre processos chamado *container shipping* é implementado para resolver o problema dos algoritmos convencionais. O *container shipping* usa o re-mapeamento de páginas entre espaços de endereçamento virtuais dos processos, movendo os dados da origem para o destino, e uma organização semi-estruturada dos dados para atingir melhoras significativas de desempenho. Os mecanismos de transferência e acesso aos dados foram desmembrados e são visíveis ao programador. O algoritmo utiliza mapeamento seletivo somente para as partes dos dados transferidos que precisam ser acessados. O *container shipping* foi concebido para a transferência de grandes objetos de dados através de um *driver* de dispositivo de entrada, de uma seqüência de processos e de um *driver* de dispositivo de saída, onde cada processo pode modificar ou não os dados transferidos

e então propagá-los em direção ao próximo processo ou *driver* de dispositivo de saída. Comparado às técnicas de transferência de dados por cópia física e por re-mapeamento de endereços virtuais, o *container shipping* superou, em muito, o desempenho das duas técnicas convencionais, pois seu custo de execução depende somente do custo de acesso aos dados efetivamente lidos ou escritos.

Druschel et al., em [9], afirmam que o SO é um fator determinante na capacidade de aproveitamento das redes de alta velocidade por parte das aplicações. É responsabilidade do SO transformar a boa vazão de rede em boa vazão entre aplicações de rede. Isto os leva a questionar se é necessário que os subsistemas de rede minimizem o número de passagens dos dados de rede através do caminho de dados entre CPU e memória, para que a largura de banda do conjunto CPU/memória das estações de trabalho permaneça dentro da mesma ordem de grandeza da largura de banda da rede. Foram pesquisadas também diversas técnicas que podem ser aplicadas a este problema. Uma importante lição, aprendida na pesquisa, é que a aplicação destas técnicas não é suficiente para atingir um bom nível de vazão entre aplicações. É necessário também integrar inteiramente o caminho de dados, desde o dispositivo origem até os programas aplicativos, passando pelo SO, para otimizar a vazão.

Druschel, em [8], avalia o ponto de contenção no tráfego de E/S do SO, com foco particular na comunicação de rede de alta velocidade. Primeiramente são identificadas as causas deste ponto de contenção, que têm suas origens na inadequação do comportamento do SO com as características de desempenho do *hardware* dos computadores modernos. A abordagem tradicional para o suporte de E/S nos SOs é novamente avaliada sob a luz do desempenho do *hardware* atual. Esta avaliação dá origem a um conjunto de novas técnicas para eliminar o ponto de contenção de E/S, que são os *fast buffers* (fbuffs) e os *application-device channels* (ADCs). Os fbuffs são usados para o gerenciamento e transferência de *buffers* de dados de E/S, através dos limites entre os domínios do usuário e do sistema. Os ADCs são uma nova facilidade do SO que toma vantagem dos protocolos de rede no nível de usuário e do suporte do adaptador de rede, dando às aplicações acesso direto, porém controlado, aos adaptadores de rede.

Smith et al., em [23], afirmam que a não adequação entre o desempenho das máquinas e a largura de banda provida pela infra-estrutura de rede, tais como comutadores e linhas de transmissão, é o fator limitante no aproveitamento da capacidade da rede. Especificamente, a largura de banda de rede é da mesma ordem de grandeza da largura de banda de memória da maioria das estações de trabalho, e por isso a carga no subsistema de memória deve ser minimizada para se obter desempenho máximo, conforme apontado por Clark et al. em [5], levando a um cuidadoso projeto da arquitetura de processamento do protocolo. A maior eficiência pode ser obtida através de diversas características de projeto, mas as principais opções são a otimização das funções de processamento na arquitetura do protocolo, a otimização do suporte ao transporte de dados do SO e uma cuidadosa colocação do *hardware* adicional necessário para o acoplamento à rede. São apresentadas abordagens para o *hardware* da interface de rede, programa de suporte da interface, e descrições das escolhas de projeto feitas na implementação de um adaptador de rede ATM para uma estação de trabalho IBM/RS6000. São analisadas, ainda, as influências sobre o desempenho das opções de projeto para as aplicações.

Dalton et al., em [6], apresentam um estudo sobre a baixa vazão efetiva detectada em algumas implementações dos protocolos TCP/IP. De acordo com o estudo, o número excessivo de cópias aos quais os dados são submetidos, desde o momento em que são gerados até o instante em que são transmitidos, é o principal responsável pela baixa vazão desses protocolos. Este excesso de cópias limitaria a vazão desses protocolos à largura de banda do sistema de memória. Como uma solução para o problema, três abordagens que permitem a redução do número de cópias são apresentadas. Uma das abordagens, chamada de cópia única, foi implementada no protocolo TCP para HP-UX. Em seguida, foram efetuadas algumas medições em estações de trabalho HP Series 700 com adaptadores de rede Afterburner, que apresentam vazão de 1Gbps. As medições mostraram que a implementação do TCP com cópia única permitia as aplicações se comunicarem numa vazão acima de 200Mbps.

Os monitores distribuídos com os SOs correntes têm a capacidade de medir a utilização de uma vasta gama de recursos do SO, incluindo tempo de CPU, a memória (RAM e os

discos), e os dispositivos de rede. Estas ferramentas fornecem dados sobre o sistema como um todo, sem discriminar a utilização de recursos por aplicação sendo executada. Um outro tipo de monitor é empregado para medir uma aplicação específica, para tal normalmente necessita a instrumentação do programa fonte da aplicação a ser monitorada e do *kernel* do SO, a pena para isso é a difícil portabilidade para outros SOs, novas versões da mesma aplicação e a restrição de uso para apenas um tipo de aplicação. Os trabalhos de avaliação de desempenho de sistemas descritos a seguir ilustram o uso de ambos os tipos de ferramentas.

Hu et al., em [11], medem e analisam o comportamento do servidor Web Apache em um sistema uniprocessado e em um sistema multi-processado simétrico com 4 processadores, com ambos os sistemas executando o SO IBM AIX. Utilizando a facilidade de rastreamento incorporada ao AIX e uma ferramenta de análise de dados, foram obtidas informações detalhadas sobre os eventos do *kernel* do SO e demais atividades do sistema, executando o Apache submetido à carga dos *benchmarks* SPECweb96 e Webstone. Descobriu-se que, em média, o Apache utiliza aproximadamente 20-25% do tempo total de CPU no domínio do usuário, 35-50% nas chamadas de sistema do *kernel* e 25-40% no tratamento de interrupções. Para sistemas com pouca memória RAM, o desempenho do servidor Web é limitado pela largura de banda do disco, pela pilha de protocolos TCP/IP e pelo tratador de interrupções de rede. Foi percebido que o Apache apresenta comportamento similar, tanto no sistema uniprocessado como no sistema multi-processado simétrico. Foram propostas 7 diferentes técnicas para o melhoramento do desempenho do Apache. Destas, 6 foram implementadas e os experimentos apresentaram uma melhora de 61% na vazão do Apache. Os resultados indicam também que o suporte do SO para o envio direto de dados do *cache* do sistema de arquivos para a rede TCP/IP pode melhorar ainda mais o desempenho do servidor Apache.

Martin et al., em [16], examinam a sensibilidade do *Network File System* (NFS) às características de desempenho de redes. Foi adotado um método de inserção de atrasos controlados em sistemas em execução para medir a sensibilidade à parâmetros básicos de rede. Foi desenvolvido um modelo de filas simples de um servidor NFS e mostrado que

este modelo caracterizava suficientemente seus dois sistemas em execução, exercitados pelo *benchmark* SPECsfs. Usando estas técnicas, é possível inferir a estrutura de servidores a partir dos resultados publicados pelo SPEC. Os resultados mostram que servidores NFS são mais sensíveis à sobrecarga do processador, podendo este ser um fator limitante. Para ganhos de desempenho em redes de alta velocidade seriam necessárias reduções na sobrecarga do processador. O NFS pode tolerar a latência de rede existente nas atuais LANS e comutadores IP. O NFS opera através de pequenas operações de meta dados e por isso é pouco sensível a largura de banda de rede. Finalmente, foi encontrado que o protocolo do NFS versão 3 tolera melhor as latências altas do que a versão 2.

Almeida, em [7], investiga a interação entre o sistema operacional e um servidor Web e apresenta os principais recursos e características do sistema que contribuem para o desempenho do servidor. Para medir a atividade e a utilização de recursos no domínio do sistema e também dos processos que compõem o servidor Web executando no domínio do usuário foi implementada a ferramenta `WebMonitor`. A ferramenta é utilizada para analisar o desempenho do servidor Apache executando sobre o Linux. Os resultados coletados mostram que grande parte do tempo de execução das requisições HTTP é gasto no domínio do sistema, isto é, executando rotinas do SO. Os resultados também mostram que o efeito de manter as conexões TCP abertas no estado TIME_WAIT, conforme especificação do protocolo TCP, causa um aumento de quase 100% no tempo de resposta necessário para tratar uma requisição HTTP, e como conseqüência uma diminuição na taxa de processamento. Uma ferramenta de *profiling* do *kernel* do Linux é utilizada para identificar os principais componentes do SO que contribuem para o tempo de execução. Os resultados mostram que grande parte do tempo é utilizado no subsistema de redes. Além disso é mostrado que as rotinas de temporização contribuem de forma significativa para a degradação do desempenho quando as conexões têm que ser mantidas no estado TIME_WAIT.

Para medição do comportamento de um sistema em um ambiente controlado onde é possível a repetição dos experimentos, bem como geração de condições extremas de operação é necessário a utilização de um gerador de carga de trabalho para controle dos

parâmetros e repetição da carga. Barford, em [3], descreve uma metodologia e uma infra-estrutura distribuída para efetuar medidas, tanto em uma rede como nos computadores a ela interligados. A primeira característica da infra-estrutura é a habilidade de gerar requisições para um servidor Web que procuram imitar o comportamento de usuários reais. Esta habilidade é baseada em uma análise detalhada do comportamento de clientes Web e da implementação do programa Gerador de Requisições URL Escalar (SURGE). O SURGE provê recursos para testar diferentes aspectos do desempenho de servidores na Web. Sua flexibilidade é demonstrada em uma avaliação das versões 1.0 e 1.1 do *Hyper Text Transfer Protocol* (HTTP). O segundo aspecto da abordagem é a análise detalhada das transações Web, pela aplicação da Análise do Caminho Crítico (ACC). A ACC permite a decomposição precisa da latência das transações Web em atrasos de propagação, variações na rede, atrasos no servidor, atrasos no cliente e atrasos por perda de pacotes. São apresentados dados de desempenho coletados pela infra-estrutura de medições.

1.3 Organização da Dissertação

Esta dissertação está organizada em capítulos dividida da seguinte forma. O capítulo 2 descreve brevemente as técnicas de avaliação de desempenho de modelagem analítica, simulação e medição, e também as ferramentas de medição de utilização de recursos do SO distribuídas com os SOs UNIX System V e Linux. O capítulo 3 descreve a ferramenta `panalyser`, seus parâmetros de configuração e sua implementação. Apresenta também a utilidade da medição de utilização de recursos do SO e compara o `panalyser` com outros monitores. O capítulo 4 apresenta um estudo de caso com medições do Apache, sendo executado em condições extremas de utilização. O capítulo 5 apresenta as conclusões sobre o trabalho aqui descrito.

CAPÍTULO 2

AVALIAÇÃO DE DESEMPENHO

Neste capítulo são apresentadas ferramentas para medição de utilização de recursos do SO distribuídas com os SOs UNIX System V e Linux em particular. A ferramenta `WebMonitor` [7], que é uma ferramenta específica à medição de desempenho do Apache, também é apresentada. São descritas algumas técnicas e enunciados alguns conceitos sobre avaliação de desempenho, para um melhor entendimento dos recursos oferecidos pelas ferramentas para medição de utilização de recursos do SO.

Conforme Jain [12], o desempenho é um critério chave no projeto, aquisição e uso de sistemas de computadores. O objetivo dos engenheiros de sistemas de computadores, cientistas, analistas e usuários é obter o melhor desempenho para um dado custo. Exemplos dos problemas envolvidos neste objetivo são: a especificação dos requerimentos de desempenho; a avaliação de alternativas de projeto; a comparação de dois ou mais sistemas; a determinação do valor ótimo de um parâmetro para o ajuste do sistema; a identificação dos pontos de contenção; a caracterização da carga no sistema; a determinação do número e tamanhos dos componentes para planejamento de capacidade; e a previsão do desempenho em cargas a serem usadas no futuro.

As medidas de vazão e tempo de resposta no transporte de dados através da rede podem apontar eventuais pontos de contenção. Pontos de contenção são os pontos do sistema com as maiores taxas de utilização. Os pontos de contenção podem ser causados pela topologia da rede, tecnologia ou capacidade do *hardware* de interligação, ou pelos computadores que compõem a rede.

Além das medidas de desempenho da rede, é fundamental a medição de parâmetros de desempenho dos computadores envolvidos. Por exemplo, podem ser tomadas medidas dos custos de processamento do SO como mediador e multiplexador do acesso de múltiplos programas aos recursos providos pelo *hardware* do sistema.

Uma vez que as medições indiquem os pontos e as condições de contenção, os administradores do sistema podem tomar as medidas apropriadas. Por exemplo, o desempenho de aplicações distribuídas pode ser afetado pelo fator de carga dos processadores na máquina, bem como pelo tráfego de entrada e saída (E/S) no disco, causado pelo uso ineficiente de memória virtual.

A avaliação de desempenho é uma tarefa trabalhosa e avaliações bem sucedidas não podem ser obtidas mecanicamente [12]. Um trabalho de avaliação de desempenho requer o conhecimento detalhado do sistema de computadores avaliado e a seleção apropriada da metodologia aplicada, carga de trabalho e ferramentas. Quando apresentados pela primeira vez, muitos problemas de desempenho não são adequadamente expressos. A maior parte do trabalho do analista é a definição e conversão do problema real em um modelo, para que ferramentas e técnicas estabelecidas possam ser utilizadas para avaliá-lo. Para auxiliar o trabalho dos analistas existem diferentes técnicas de avaliação de desempenho, as quais serão brevemente descritas a seguir.

2.1 Técnicas de Avaliação de Desempenho

As três principais técnicas para avaliação de desempenho são *modelagem analítica*, *simulação* e *medição* [12]. A modelagem analítica é baseada na simplificação do sistema em um modelo que reflete apenas os aspectos essenciais do seu comportamento. A modelagem provê suporte para reunir, organizar, avaliar e entender informações sobre um sistema de computadores [15]. Uma vez que um modelo tenha sido definido através deste processo de abstração, ele pode ser parametrizado para refletir alternativas sob estudo, e então, estas podem ser avaliadas. Usar um modelo para investigar o comportamento de um sistema é menos trabalhoso e mais flexível do que a medição, porque detalhes desnecessários são desconsiderados.

A simulação é um meio de imitar o comportamento de um sistema sob certos aspectos. No sentido técnico da palavra, simulação é uma metodologia que usa modelos para o estudo do sistema existente ou postulado. O modelo para simulação pode ser desde um conjunto de equações até um programa. O simulador que melhor se adapta a uma

determinada situação depende de quão fiel é a representação do sistema modelado e dos pré-requisitos e objetivos da simulação. Um simulador oferece um modo simples de prever o desempenho ou comparar diversas alternativas para um sistema. Um modelo de simulação pode ser preferido à medições, porque permite que as alternativas sejam comparadas sob uma gama de cargas e ambientes mais ampla do que em medições [17].

A medição de sistemas de computadores envolve a monitoração dos mesmos, enquanto submetidos a cargas de trabalho específicas. Para a obtenção de resultados razoáveis, as cargas de trabalho devem ser cuidadosamente escolhidas. A medição somente é possível com a existência de um sistema real a ser medido, bem como de ferramentas para realizar as medições [10, 12].

Um monitor é uma ferramenta usada para observar a atividade de um sistema. Em geral, monitores observam o desempenho dos sistemas, coletam estatísticas sobre o desempenho, analisam os dados e mostram resultados. Alguns monitores também identificam áreas problemáticas e sugerem correções. Monitores são usados não somente por analistas de desempenho, mas também por programadores e gerentes de sistema para: encontrar segmentos do programa freqüentemente usados e otimizar seu desempenho; para medir alocação de recursos e encontrar pontos de contenção no desempenho; para fazer o ajuste do sistema e melhorar o desempenho; para caracterizar uma carga de trabalho, para planejamento de capacidade e criação de cargas de trabalho de testes; e para encontrar os parâmetros de um modelo, validar modelos, e desenvolver dados de entrada para modelos [12].

Os monitores utilizam métricas de desempenho para a apresentação dos resultados das medições realizadas. Estas métricas serão brevemente descritas a seguir.

2.2 Métricas de Desempenho

Métricas de desempenho são grandezas dimensionais que refletem a *capacidade de processamento*, o número de *requisições de serviço*, a *vazão*, o *tempo de resposta* e a *eficiência* dos sistemas de computadores [12]. A capacidade de processamento é definida como a taxa pela qual as requisições de serviço podem ser atendidas pelo sistema, medida em

requisições por unidade de tempo. Para sistemas de batelada a capacidade de processamento é medida, por exemplo, em *jobs* por segundo (job/s). Para sistemas interativos a capacidade de processamento é medida em requisições por segundo (req/s). Para unidades centrais de processamento (CPU) a capacidade de processamento é medida em milhões de instruções por segundo (MIPS), ou milhões de operações ponto flutuante por segundo (MFLOPS). Para o processamento de transações de banco de dados, por exemplo, a capacidade de processamento é medida em transações por segundo (tps).

O número de requisições de serviço podem ser definidas como transações de banco de dados, *jobs* em sistemas de batelada, operações de E/S, operações de programa e instruções de programa, dependendo do sistema em foco.

A vazão é definida como o número de unidades de informação que atravessa um meio físico ou interface, sendo medida em pacotes por segundo (pps), ou bits ou bytes por segundo (bps ou Bps).

O tempo de resposta pode ser definido de várias formas. Por exemplo, como o intervalo decorrido entre o fim da submissão de uma requisição e o final da resposta correspondente do sistema.

A capacidade de processamento ou a vazão de um sistema inicialmente cresce com o aumento da carga no sistema. Acima de uma certa carga, a capacidade de processamento/vazão deixa de crescer e na maioria dos casos pode começar a decrescer, conforme demonstrado no primeiro gráfico da Figura 2.1. A capacidade de processamento máxima sob condições de trabalho ideais é chamada *capacidade nominal* do sistema. Para redes de computadores, a capacidade nominal é chamada *largura de banda* e é usualmente expressa na unidade de vazão (bps). Em uma rede, por exemplo, o tempo de resposta sob regime de vazão máxima pode ser alto demais para ser aceitável. Neste caso, é interessante conhecer a vazão máxima que pode ser atingida, sem exceder um limite de tempo de resposta pré-especificado. Isto pode ser chamado de *vazão utilizável* da rede. A denominação análoga para um sistema é *capacidade de processamento utilizável.* Em muitas aplicações, o ponto de inflexão das curvas de capacidade de processamento ou vazão e tempo de resposta é considerado o ponto de operação ótimo. Este é o ponto além do

qual o tempo de resposta aumenta rapidamente em função da carga, porém o ganho de capacidade de processamento/vazão é pequeno. Abaixo do ponto de inflexão, o tempo de resposta não aumenta significativamente, mas a capacidade de processamento/vazão cresce com o aumento da carga.

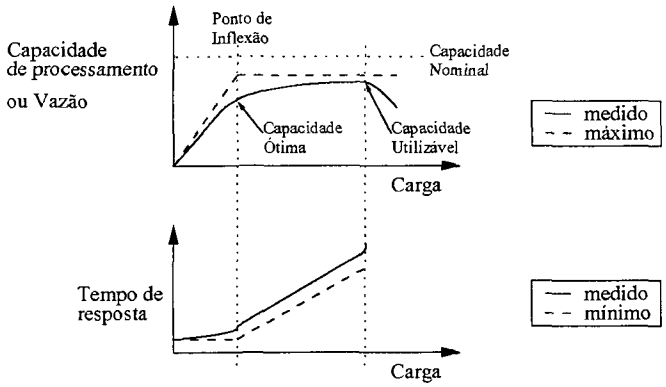

Figura 2.1: Capacidade de um sistema.

A relação entre a capacidade de processamento utilizável e capacidade nominal ou vazão utilizável e largura de banda é chamada eficiência. Por exemplo, se a vazão que pode ser utilizada de uma rede local de 100Mbps é somente 60Mbps, a sua eficiência é portanto de 60%.

Ao se aumentar a capacidade nos pontos do sistema onde o regime de operação atingiu a capacidade de processamento/vazão máxima possível com um tempo de resposta aceitável, melhora-se o desempenho do sistema como um todo. Introduzir melhorias em outros pontos do sistema têm baixa relação de custo/benefício, conforme lei de Amdahl [10].

A próxima seção apresenta alguns conceitos de SO e discute porque o desempenho do SO propriamente dito é importante para o desempenho das aplicações que são executadas sobre ele. Também serão descritas algumas ferramentas de medição de utilização de recursos do SO.

2.3 Desempenho em Sistemas Operacionais

O papel de um Sistema Operacional é mediar e multiplexar o acesso de múltiplos processos aos recursos providos pelo *hardware* [2, 24]. Em um SO da família UNIX os *programas* são executados na forma de *processos*.

Um processo pode ser definido como um programa em execução. O processo é uma entidade dinâmica, mudando constantemente à medida que as instruções do programa são executadas pelo processador [21, 2, 24]. Além das instruções e dados de um programa, o processo também inclui o *contador de programa*, todos os registradores da CPU e as pilhas do processo contendo dados temporários tais como parâmetros de rotina, endereços de retorno e variáveis temporárias armazenadas. O processo corrente inclui toda a atividade do processador. Cada processo é executado em seu próprio espaço de endereçamento, e não pode interagir com outros processos exceto através de mecanismos seguros, gerenciados pelo *kernel* do SO. Um programa é um conjunto de instruções em código de máquina (e dados) armazenados em uma imagem executável.

As funções do SO podem ser caracterizadas em duas seções ou domínios, a *seção de E/S no espaço do usuário* e a seção do *kernel*. A seção de E/S no espaço do usuário consiste de rotinas de biblioteca, que são ligadas aos programas de usuário, e de programas inteiros executados fora do *kernel*. A maior parte do subsistema de E/S está dentro do *kernel* do SO. As chamadas de sistema, incluindo as chamadas para E/S, normalmente são executadas por procedimentos de biblioteca.

O *kernel* é a seção responsável pela mediação entre as operações de E/S solicitadas pelos processos de usuário e o *hardware*, dentre outras funções. Pela construção estratificada do *kernel*, os pontos de E/S do mesmo encontram-se na comunicação entre seus níveis, caracterizando-o portanto, como um *subsistema de E/S*, dada a sua função de interface entre os programas e o *hardware* [6, 8, 9]. O *kernel* pode ser dividido em três níveis [2, 4, 24], quais sejam, (i) *rotinas de tratamento de interrupção*, (ii) *drivers de dispositivos* e (iii) *nível independente de dispositivos*. Os níveis são mostrados na Figura 2.2.

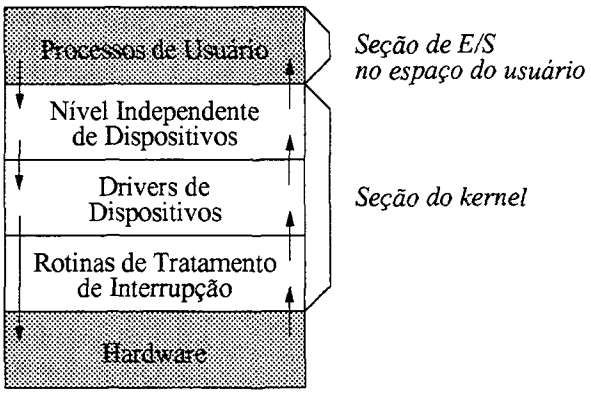

Figura 2.2: Níveis de divisão do subsistema de E/S do *kernel*.

As interrupções geradas pelo *hardware* devem ser tratadas pelos níveis mais baixos do modelo estratificado do SO, de forma que o mínimo possível de partes do sistema tenha que interagir com elas. A melhor maneira de esconder o mecanismo de interrupções é fazer com que todo o processo que inicie uma operação de E/S fique bloqueado até o término de tal operação, fato este que é sinalizado através de uma interrupção. O processo pode auto bloquear-se executando, por exemplo, um *DOWN* em um semáforo, ou um *WAIT* em uma variável de condição, ou um *RECEIVE* em uma mensagem. Quando a interrupção ocorrer, a rotina de tratamento da interrupção faz o que for preciso para desbloquear o processo que iniciou a operação.

Todo o código que depende do dispositivo físico está no *driver* do dispositivo. Cada *driver* trata um tipo de dispositivo, ou no máximo, uma classe de dispositivos muito semelhantes. As controladoras de sistema de E/S no *hardware*, normalmente tem um ou mais registradores de dispositivo, usados para enviar os comandos a serem executados pela controladora. O driver do dispositivo gera tais comandos e verifica se eles foram corretamente executados. Em termos gerais, o trabalho de um driver de dispositivo é o de aceitar requisições de alto nível por parte do programa independente de dispositivo, e fazer com que tal requisição seja atendida. Uma requisição típica é um pedido para leitura do bloco N do disco, por exemplo, caso o driver esteja livre no momento da chegada da requisição, este começa a executá-la imediatamente. Caso esteja ocupado tratando de outra requisição, a recém-chegada será colocada em uma fila de espera.

A maior parte do subsistema de E/S é independente dos dispositivos. O limite exato

entre os *drivers* e o nível independente dos dispositivos varia em cada projeto de SO, pois algumas funções que podem ser realizadas de forma independente do dispositivo podem ser, também, realizadas no *driver* por motivos de eficiência ou projeto. O objetivo básico do nível independente de dispositivos é a realização de funções de E/S que são comuns a todos os dispositivos, oferecendo uma interface uniforme para o nível de programas de usuário. Tipicamente as funções deste nível são: interface uniforme para os drivers de dispositivo; identificação do dispositivo; proteção do dispositivo; fornecimento de um tamanho de bloco, independente do dispositivo; *buffering*; alocação de espaço para blocos; alocação e liberação de dispositivos dedicados; informação de erro.

Os pontos de contenção, relacionados a E/S nos SOs, têm motivado os pesquisadores a reavaliar as soluções tradicionais para suportar operações de E/S, sob a luz do desempenho dos atuais dispositivos de *hardware*. Tais pontos de contenção têm suas origens em funcionalidades do SO, ou suas implementações, não adequados às características de desempenho do *hardware* dos computadores modernos [8]. Pasquale et al., em [19], afirmam que a maioria dos SOs não suporta adequadamente aplicações que geram tráfego intensivo de E/S, por não empregarem algoritmos eficientes de transferência de dados entre os processos que compõem estas aplicações, e entre processos e o sistema de arquivos, protocolos de rede e servidores de janelas. A estrutura computacional dinâmica formada por estas iterações é tipicamente um duto de E/S, que pode ser descrito como uma atividade repetitiva, onde um processo carrega um quantidade de dados, estes dados são seqüencialmente transferidos entre processos, cada qual podendo ler ou modificar parte dos dados, e um dos processos descarrega os dados possivelmente modificados. Processos que executam código específico da aplicação podem requerer acesso a todos os dados ou a parte deles. Entretanto, processos que controlam a transferência a partir do dispositivo de E/S por acesso direto a memória (DMA), tipicamente não necessitam de acesso aos dados ou apenas precisam acessar uma pequena porção dos dados. A maioria dos SOs é ineficiente na transferência de dados entre os espaços de endereçamento dos processos. O UNIX, por exemplo, requer a cópia física dos dados entre os domínios de proteção do *kernel* e dos processos de usuários [2]. A cópia física prejudica o desempenho do SO e dos

programas relacionados ao sistema. Isto é mais evidente nas implementações do programa de protocolo de rede, onde a cópia física pode utilizar uma fração significativa do tempo de processamento para grandes pacotes de dados [9].

A Figura 2.3 mostra como a cópia física entre os domínios do processo e do *kernel* ocorre durante operações de E/S no UNIX, levando a degradações no desempenho para grandes transferências de dados. A vazão total é limitada pela taxa de cópia dos dados que não estão em *cache*, mostrada na figura como a menor largura de caminho de dados. Este é o ponto de contenção de todo o duto de E/S. A ineficiência é piorada pelo tempo despendido nas transferências para copiar os dados entre vários domínios distintos.

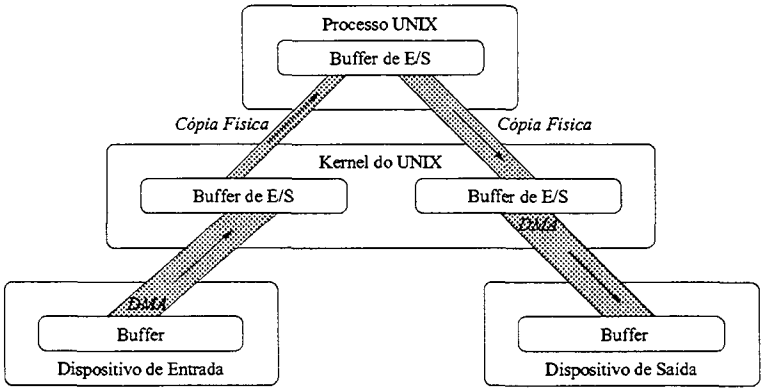

Figura 2.3: Cópia física em um *pipeline* de E/S simples no UNIX.

2.4 Ferramentas de Avaliação de Desempenho

Dentre as ferramentas de avaliação de desempenho distribuídas com os SOs correntes destacam-se as ferramentas `strace`, `atsar`, `ps`, `top`, presentes nos SO UNIX comerciais baseados no padrão System V, e no Linux, que são descritas abaixo. Também é descrita a ferramenta `WebMonitor`, usada para avaliação de desempenho do programa servidor Apache no Linux.

2.4.1 Strace

A ferramenta `strace` intercepta e registra as chamadas de sistema efetuadas pelo processo rastreado, bem como os sinais que são recebidos pelo processo. O nome de cada

chamada de sistema e seus argumentos são impressos na saída de erro padrão ou em um arquivo. É possível gerar um sumário ao término da execução do programa rastreado, ou na interrupção do rastreamento, contendo o tempo decorrido, a quantidade e os erros para cada uma das classes de chamadas de sistema efetuadas [1].

2.4.2 Ferramentas Baseadas no /proc

As ferramentas `atsar`, `free`, `ps` e `top`, são baseadas no pseudo sistema de arquivos `/proc` [25]. O `/proc` é usado como uma interface para as estruturas de dados do *kernel*, e para leitura e interpretação de `/dev/kmem`, que é o arquivo associado ao dispositivo que representa a memória virtual do *kernel* do SO Linux [27]. É importante ressaltar que embora o `/proc` ofereça uma ampla variedade de informações, a sua utilização freqüente causa um alto impacto no SO. A ineficiência da utilização do `/proc` é causada pela necessidade de realizar várias leituras no sistema de arquivos ou ler grandes quantidades de dados. O alto custo de acesso ao sistema de arquivos se deve ao fato do processo em questão interromper sua execução e aguardar uma resposta indicando se a leitura obteve sucesso ou não. Enquanto o processo espera, o escalonador pode escolher um outro processo para executar. Se a freqüência com que o processo em questão efetua chamadas, ou se o tamanho da requisição no *read* for grande, isto acarretará desperdício do tempo de CPU em atividade de E/S.

Segue uma descrição breve de funcionalidades e uso das ferramentas. O **atsar** [14] é um coletor de dados que lê os contadores de eventos do *kernel* através do pseudo-sistema de arquivos `/proc`. O `atsar` apresenta e formata os dados dos contadores e os apresenta na saída padrão. Este programa pode ser comparado aos programas padrão `sadc` e `sar` fornecidos com outras implementações do UNIX.

Os dados fornecidos pelo `atsar` compreendem estatísticas sobre utilização de CPU, de discos, de memória, de *swap*, das tabelas do *kernel* e dos ttys, bem como o número de interrupções sofridas pelo processador. São fornecidos também dados sobre o tráfego e os erros das interfaces de rede e sobre os protocolos IP, ICMP, TCP e UDP. Ainda são informados pelo `atsar` a utilização do NFS e opcionalmente o tráfego FTP e HTTP.

O **ps** fornece informações sobre o status dos processos em execução tais como tempo de execução e estado, por exemplo. A descrição das informações que podem ser exibidas está no manual da ferramenta [13] e na documentação do `/proc` [25]. O funcionamento do ps resume-se à leitura dos arquivos `/proc/<número do processo>/stat` e na formatação da saída.

O **top** apresenta uma listagem periódica das atividades de processamento e lista as tarefas que têm a maior utilização de CPU. A amostra dos processos pode ser ordenada por utilização de memória e por tempo de execução. Facilidades para o controle e exibição do estado dos processos podem ser selecionados por meio de uma interface interativa ou através da especificação em arquivo de configuração. A descrição das informações que podem ser exibidas está no manual da ferramenta [22] e na documentação do `/proc` [25].

2.4.3 WebMonitor

O `WebMonitor` é uma ferramenta desenvolvida por Almeida e outros [7], que emprega uma combinação de técnicas de monitoração por amostragem e técnicas de monitoração orientada por eventos para coletar diferentes níveis de informação sobre a operação de um servidor Web `Apache` versão 1.1.1 [20].

A coleta baseada em amostragem é usada para ler os valores de contadores mantidos pelo *kernel*. Estes contadores representam informações no nível do sistema, tais como utilização de recursos, taxas de interrupção e também estatísticas de utilização da rede. O `WebMonitor` permite intervalos diferentes entre coletas, dependendo da natureza da informação coletada. Por exemplo, devido ao grande número de conexões TCP em um servidor sobrecarregado, pode ser interessante monitorar a porta de comunicação freqüentemente. Por outro lado, informações sobre o sistema como um todo, tal qual a atividade do disco, podem ser monitorados com uma freqüência menor em um mesmo período de tempo. Para tomar amostras de uma maneira mais eficiente que a leitura do pseudo sistema de arquivos `/proc` foram necessárias modificações no *kernel* do Linux.

Para o monitoramento das requisições HTTP emprega-se a técnica de monitoração orientada por eventos, por ser mais adequada ao perfil de execução das requisições HTTP.

O tratamento de uma requisição pode ser dividido em fases seqüenciais e distintas, o que torna sua monitoração controlada pela ocorrência de eventos mais adequada. Para implementar a técnica de monitoração orientada a eventos, foi necessária a instrumentação do código do programa servidor Apache. As instrumentações do *kernel* e do Apache, usadas para viabilizar a implementação `WebMonitor`, permitem uma grande precisão nas medições.

CAPÍTULO 3

A FERRAMENTA PANALYSER

As informações sobre a utilização de recursos do SO, discriminadas por processo, permitem quantificação precisa da utilização de recursos do SO pelos processos analisados. Estas informações permitem ampliar a gama de fatores para determinação da melhor configuração, otimização e ampliação de um sistema para o atendimento dos serviços para os quais foi concebido.

O `panalyser` é um monitor que opera em modo batelada, sendo controlado por eventos e amostragem, e que permite medir a utilização de recursos pelos processos no SO Linux. As informações apresentadas possibilitam desenvolver e avaliar o desempenho de aplicações, diagnosticar pontos de contenção no sistema, caracterizar cargas de trabalho, fornecer parâmetros e validar modelos.

A seguir são descritos os dados fornecidos pelo `panalyser` e seus parâmetros de configuração. São também tecidas considerações sobre sua implementação e apresentada uma comparação entre resultados de medições sobre um programa de teste simplificado de comportamento determinístico, fornecidas pelo `panalyser`, `atsar` e `strace`.

3.1 Informações Fornecidas pelo Panalyser

As informações de utilização de recursos do SO, fornecidas pelo `panalyser`, são baseadas na estrutura de dados `rusage`, a qual é preenchida pela chamada de sistema `getrusage` [26]. Uma *chamada de sistema* é o conjunto de instruções que estabelece a interface entre o *kernel* e o programa de usuário [2]. Um dos fatores que contribuem para baixo custo de execução do `panalyser` é o uso da chamada de sistema `getrusage` ao invés da leitura constante do `/proc` para obtenção de dados. Por uma limitação de implementação do *kernel* 2.4.16 do Linux apenas os campos `ru_utime`, `ru_stime`, `ru_minflt`, `ru_majfl` e `ru_nswap` são preenchidos. A estrutura de dados `rusage` é apresentada na

Figura 3.1

```
struct rusage {
---->struct timeval ru_utime; /* user time used */
---->struct timeval ru_stime; /* system time used */
     long ru_maxrss;          /* maximum resident set size */
     long ru_ixrss;           /* integral shared memory size */
     long ru_idrss;           /* integral unshared data size */
     long ru_isrss;           /* integral unshared stack size */
---->long ru_minflt;          /* page reclaims */
---->long ru_majflt;          /* page faults */
---->long ru_nswap;           /* swaps */
     long ru_inblock;         /* block input operations */
     long ru_oublock;         /* block output operations */
     long ru_msgsnd;          /* messages sent */
     long ru_msgrcv;          /* messages received */
     long ru_nsignals;        /* signals received */
     long ru_nvcsw;           /* voluntary context switches */
     long ru_nivcsw;          /* involuntary context switches */
};
```

Figura 3.1: A estrutura de dados `rusage`. As setas indicam os campos utilizados na versão 2.4.16 do *kernel* do Linux.

Os dados de utilização de tempo da CPU no domínio de usuário, `ru_utime`, e no domínio do sistema operacional, `ru_stime`, juntamente com a informação de tempo real decorrido, permitem ao `panalyser` fornecer a utilização da CPU. O dado de utilização de tempo da CPU no domínio do SO não inclui o tempo de tratamento de interrupções do processador.

O `ru_minflt` fornece o número de páginas de memória requisitadas pelo processo que não foram lidas do disco. O `ru_majlt` fornece o número de páginas de memória requisitadas pelo processo que foram lidas do disco. O `ru_nswap` fornece o número de operações de `swap` realizadas. Estes dados demonstram a utilização de memória primária e secundária pelo processo analisado.

É importante observar que os campos não preenchidos na estrutura `rusage` conteriam dados muito importantes sobre a utilização de recursos do SO, tais como: uso da memória primária, leitura e escrita de blocos, troca de mensagens, número de sinais recebidos e trocas de contexto. A implementação do `panalyser` contempla estes campos da estrutura, ou seja, quando seu preenchimento for implementado no *kernel*, estas informações serão

uso propostos.

As chamadas de sistema, efetuadas pelo processo analisado, são classificadas e contabilizadas. Estes dados podem servir para determinar o volume de chamadas de sistema de cada classe ou tipo, identificando também se chamadas de sistema de alto custo estão sendo invocadas muito freqüentemente. Estas informações podem ser usadas na depuração e otimização de programas.

Em conjunto com os dados mencionados são apresentados ainda a média, o desvio padrão, o coeficiente de variação e a mediana de séries temporais de medidas. Também são fornecidos, no início e no final da análise, dados sobre a utilização de memória virtual e o conjunto de páginas residentes do processo analisado. A comparação destes dados auxilia a identificar eventuais anomalias na utilização da memória pelo processo analisado.

3.2 Parâmetros do Panalyser

O `panalyser` é configurado através da linha de comando. Para seu uso é necessário fornecer o número do processo a ser rastreado, através do parâmetro `-p` ou o nome do programa a ser executado e rastreado, bem como os parâmetros para geração das informações, sobre a utilização de recursos do SO, como o intervalo entre amostragens, tempo de medição, se os processos filhos serão rastreados e em qual intervalo de ocorrência, e se serão gerados dados estatísticos sobre a utilização de recursos.

É possível determinar o intervalo entre amostragens efetuadas pelo `panalyser` com precisão de segundos e microsegundos, através dos parâmetros `-i` e `-u`, respectivamente. A duração da medição pode ser determinada pelo fim do processo rastreado, indicado pelo parâmetro `-e`, ou por tempo com a precisão de minutos e segundos, através dos parâmetros `-T` e `-t`, respectivamente. É possível ainda determinar quais processos filhos

temporais de medidas, a média, a variância, o desvio padrão, a mediana e o coeficiente de variação. O cálculo da mediana depende da escolha efetuada no parâmetro `-c` que estabelece o critério de ordenação dos dados, dentre o número de chamadas de sistema, o tempo no domínio do usuário ou o tempo no domínio do sistema. Os dados normalmente são escritos na saída padrão, mas podem ser colocados em um arquivo definido pelo parâmetro `-o file`.

Os parâmetros de configuração do `panalyser`, passados na linha de comando, são listados sempre que a ferramenta é acionada sem parâmetros ou com a opção `-h`. Os parâmetros são mostrados na Figura 3.2.

```
usage: panalyser [-d] [-f] [-h] [-e] [-i interval] [-u interval] [-t seconds]
[-T minutes] [-s [-c ord criteria]] [-m file] [-o file]... [-p pid]
or: panalyser ... [command [arg ...]]
-d -- print debug messages to stderr
-f interval_fforks -- follow forks of process with the specified interval
between number of forks, creating log files with name FILE.PID
-h -- print this help message
-e -- analyse the process until it ends. [-t -T] are ignored
-i seconds -- seconds between samples
-u useconds -- microseconds between samples
-t seconds -- duration of measurement in seconds
-T minutes -- duration of measurement in minutes
-s -- enable statistics (mean, variance, std deviation, median and
coef. of variation) this option creates files with tabular data
with names FILE_rusage.dat and FILE_syscall.dat
-c order criteria -- for median, can be one of: NSYSCALLS, UTIME, STIME(default)
-m file -- temporary FILE name
-o file -- send trace output to FILE instead of stderr
-p pid -- trace process with process id PID
```

Figura 3.2: Parâmetros de configuração.

3.3 Considerações Sobre a Implementação

A implementação do `panalyser` se baseia nas chamadas de sistema `ptrace`, `wait4` e

para Linux, sendo portável para todas as plataformas de *hardware* suportadas por este SO.

A chamada de sistema `ptrace` provê meios para que um processo possa observar e controlar a execução, e examinar e mudar a imagem e registradores de outro processo. Esta chamada é usada principalmente para implementar pontos de verificação e acompanhamento de chamadas de sistema em um processo rastreado por outro.

A chamada de sistema `wait4` suspende a execução do processo corrente até que um processo filho, especificado nos argumentos de chamada da `wait4`, tenha terminado ou até que um sinal tenha sido entregue ao último.

A chamada de sistema `getrusage` retorna a utilização corrente de recursos de um processo, segundo a estrutura `rusage`, descrita em [26] e na Seção 3.1.

Os dados coletados são armazenados em memória temporária, sendo escritos no disco para esvaziar a memória temporária, quando esta está cheia.

O `panalyser` foi projetado de forma a permitir mudanças na implementação para permitir o rastreamento de mais de uma geração de processos sem um limite determinado. A atual implementação foi limitada a uma geração de filhos do processo rastreado, pois a arquitetura da maioria das aplicações que geram filhos é baseada em um processo principal que recebe as requisições e gera filhos para atendê-las.

Uma versão esquemática do algoritmo de funcionamento do `panalyser` é apresentado na Figura 3.3. As etapas da execução do algoritmo de monitoração e análise são descritas a seguir:

1. Tratamento dos parâmetros de entrada.

2. Abertura dos arquivos de saída. Caso a geração de dados estatísticos tenha sido habilitada, serão abertos arquivos separados para a saída formatada dos dados da

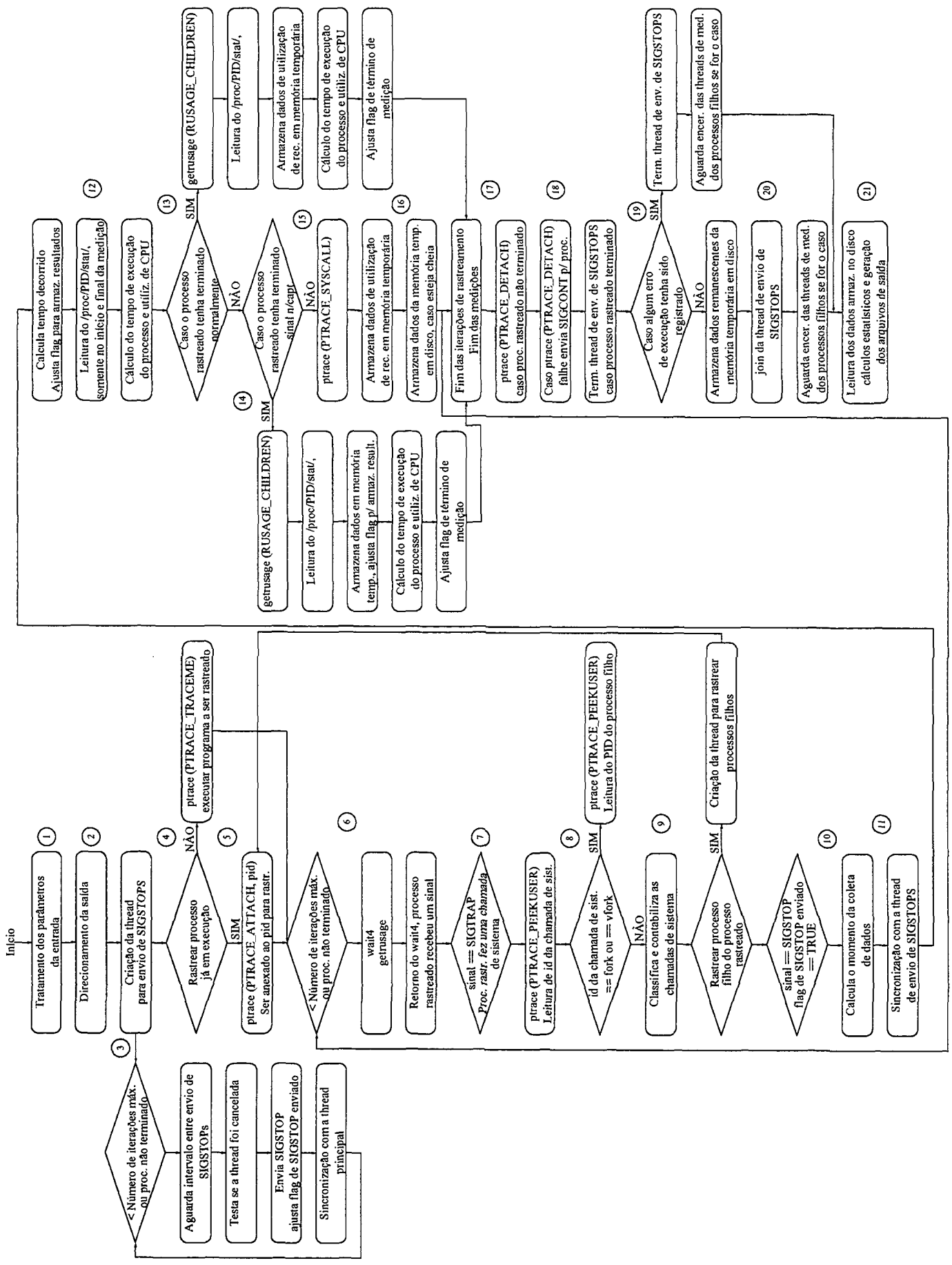

Figura 3.3: Algoritmo de funcionamento do `panalyser`.

rastreado no intervalo determinado pelos parâmetros de entrada. Antes de enviar um SIGSTOP a *thread* verifica se recebeu um sinal de cancelamento, evitando assim o envio de um sinal de SIGSTOP para o processo rastreado (o qual não seria seguido do respectivo SIGCONT, provocando a parada do processo rastreado por tempo indeterminado). Após o envio do SIGSTOP o *flag* de SIGSTOP enviado é ajustado para permitir a correta execução do processo principal do panalyser. Depois de enviar o SIGSTOP a *thread* ficará aguardando o aviso de sincronização do processo principal, para enviar outro SIGSTOP, somente quando o processo principal estiver apto a tratá-lo. Desta forma, o intervalo de amostragens pode ser alongado pelo tempo necessário ao processamento do SIGSTOP anterior pelo processo principal.

4. Dependendo dos parâmetros de configuração do panalyser, o processo a ser rastreado será iniciado pelo próprio panalyser, ou o panalyser se anexará a um processo iniciado anteriormente. No primeiro caso o panalyser fará uma chamada de sistema fork para criar o novo processo que executará o programa a ser rastreado. A seguir fará uma chamada de sistema ptrace, com o parâmetro (PTRACE_TRACEME), a qual indicará que este processo terá sua execução rastreada pelo panalyser. A partir desta chamada de sistema, quaisquer sinais entregues ao processo rastreado, exceto SIGKILL, causarão a parada e notificação do panalyser quando este fizer uma chamada de sistema wait4. Todas as chamadas de sistema execve subseqüentes, efetuadas pelo processo rastreado, causarão o envio de um sinal SIGTRAP para o próprio processo, causando sua parada e notificação do panalyser quando este fizer uma chamada de sistema wait4. A seguir é efetuada uma chamada de sistema execve para executar o programa que será rastreado.

5. No segundo caso, o panalyser rastreará um processo já existente. Para isso executará uma chamada de sistema ptrace com os parâmetros (PTRACE_ATTACH, pid). O pid é o identificador do processo (PID) que será rastreado. A partir desta chamada de sistema, o comportamento do processo rastreado será igual ao descrito para a chamada ptrace com o parâmetro PTRACE_TRACEME no item 4.

6. A partir deste ponto, o laço de iterações para medição do processo rastreado é iniciado e será repetido em todos os sinais recebidos pelo processo rastreado, até que o número de medições válidas seja atingido ou até que o processo rastreado termine. Quando o processo rastreado recebe um sinal no processo que executa o `panalyser` o `wait4` retorna com os campos da estrutura `rusage` preenchidos pelo SO.

7. Para registrar as chamadas de sistema, executadas pelo processo rastreado, é necessário invocar a `ptrace` com os parâmetros `(PTRACE_SYSCALL, pid)`, que fará com que o processo rastreado, identificado por `pid`, continue sua execução e receba um sinal `SIGTRAP` sempre que efetuar uma chamada de sistema, provocando então sua parada e a sinalização do `panalyser`.

8. Para identificar qual foi a chamada de sistema executada pelo processo rastreado, o `panalyser` efetua uma chamada de sistema `ptrace` com os parâmetros `(PTRACE_PEEKUSER, add)`, que retorna o identificador da chamada de sistema lido no endereço `add`. Caso a chamada de sistema seja um `fork` ou `vfork`, é efetuada a leitura do identificador do processo filho do processo rastreado, com nova chamada de sistema `ptrace`.

9. A chamada de sistema, executada pelo processo rastreado, é classificada e contabilizada. Caso o processo filho do processo rastreado seja rastreado, será criada uma *thread* para rastrear o novo processo. O algoritmo da nova *thread* é aquele do item 5.

10. Caso a sinalização do recebimento de um `SIGSTOP`, pelo processo rastreado tenha ocorrido e o *flag* de `SIGSTOP` enviado tenha sido ajustado pela *thread* de envio de `SIGSTOPs`, a amostragem de dados de utilização de recursos é válida e então é calculado o momento da coleta dos dados.

11. Aqui o `panalyser` emite uma sinalização de sincronização para a *thread* de envio de `SIGSTOPs`. O próximo `SIGSTOP` pode então ser enviado, pois o processo principal já está apto a tratá-lo.

12. Neste ponto é calculado o tempo real decorrido. O *flag* para armazenar resultados

é ajustado, alertando que existem medições armazenadas na memória temporária. Caso seja o início ou o final da medição, o arquivo `/proc/PID/stat` é lido para recuperar os dados de utilização de memória do processo rastreado, que ainda não foram implementados na chamada de sistema `getrusage` na versão do *kernel* do Linux utilizada neste trabalho. Também é calculado o tempo de execução e a utilização de CPU do processo rastreado.

13. Caso o processo rastreado tenha terminado normalmente, uma chamada de sistema `getrusage` é efetuada para recuperar os recursos utilizados pelo processo ao final de sua execução. O arquivo `/proc/PID/stat` é lido para recuperar os dados de utilização de memória do processo rastreado. O valor final para os dados de utilização de recursos são armazenados em memória temporária. O valor final para a utilização de CPU pelo processo rastreado e seu tempo de execução são calculados. O *flag* de término de medição é ajustado e o algoritmo é desviado para o passo descrito no item 17.

14. Caso o processo rastreado tenha terminado por um sinal não capturado, procede-se a geração dos dados finais de medição e o algoritmo é desviado para o passo descrito no item 17.

15. É efetuada a chamada de sistema `ptrace` com os parâmetros `(PTRACE_SYSCALL, pid)`, para que o processo rastreado continue sua execução e receba um sinal `SIGTRAP` sempre que fizer uma chamada de sistema.

16. Os dados de utilização de recursos do processo rastreado são armazenados em memória temporária. Caso a memória temporária esteja cheia, os dados são armazenados em disco. Retorna ao passo descrito no item 6.

17. Este é o ponto, logo após o término do laço de iterações de medição, a partir de onde serão executados os passos de término do algoritmo.

18. Caso o processo rastreado não tenha terminado, é efetuada uma chamada de sistema `ptrace` com o parâmetro `(PTRACE_DETACH)` para "desligar" o `panalyser` do processo

rastreado e permitir que o processo rastreado continue sua execução normalmente. Caso a chamada de sistema `panalyser (PTRACE_DETACH)` falhe, um sinal `SIGCONT` é enviado ao processo rastreado para não interromper sua execução normal, após o término da medição. Caso o processo rastreado tenha sido terminado, a *thread* de envio de `SIGSTOPs` é sinalizada para terminar sua própria execução.

19. Caso ocorra algum erro durante as iterações de medição, serão executados os procedimentos de finalização do `panalyser` e o algoritmo será desviado para o passo do item 21.

20. Os dados remanescentes na memória temporária são armazenados em disco e efetuam-se os procedimentos de finalização do `panalyser`.

21. No final da execução do `panalyser` é efetuada a leitura dos dados da medição armazenados no disco, os cálculos estatísticos e a geração dos arquivos de saída.

3.4 Comparação com Outros Monitores

Dentre as ferramentas para medição de utilização de recursos do sistema operacional, destacam-se o `atsar` pela quantidade de informações fornecidas sobre a utilização dos recursos do SO, e o `strace` pelas informações sobre as chamadas de sistema efetuadas por um processo.

A Tabela 3.1 apresenta uma comparação do tempo de execução, nos domínios do usuário e sistema, de um programa de teste de comportamento determinístico, sem ser rastreado e sendo rastreado pelo `atsar`, `strace` e `panalyser`. O objetivo desta medição é comparar o grau de intrusão do `panalyser` com o das outras ferramentas.

Tempo de Execução (s)	Condições da Medição						
	não rastr.	`atsar`	var. %	`strace`	var. %	`panalyser`	var. %
domínio do usuário	4,4096	4,4160	0,15	7,5594	41,67	7,1756	38,55
domínio do sistema	13,4436	13,7330	2,11	22,3054	39,73	21,3514	37,04

Tabela 3.1: Tempo de execução nos domínios do usuário e sistema do programa de teste não rastreamento e sendo rastreado pelo `atsar`, `panalyser` e `strace`.

O `panalyser` expande o tempo de execução do programa de teste em 38,55 % para o domínio do usuário e 37,04 % no domínio do sistema. Isto pode ser explicado pelo grande número de chamadas de sistema efetuadas pelo programa de teste. Como o `panalyser` executa a amostragem de chamadas de sistema por evento, o grande número de eventos faz com que o algoritmo do `panalyser` seja repetido para cada evento, interferindo significativamente na execução do programa de teste. Como o `atsar` não faz amostragens das chamadas de sistema por processo ele apresenta uma menor intrusão uma vez que efetua amostragens apenas em intervalos determinados sem atender a eventos como no caso do `panalyser` e `strace`. O `strace` apresenta uma intrusão semelhante à apresentada pelo `panalyser`, pois também faz o rastreamento das chamadas de sistema.

Os experimentos foram realizados em uma computador com processador Intel Pentium III a 800MHz com *cache* de 256KB integrada, 192MB de memória SDRAM a 100MHz, executando Debian Linux 3.0, com *kernel* Linux 2.4.16 i686.

Este programa de teste foi desenvolvido em C padrão ANSI e compilado pelo compilador `gcc version 2:2.95.4-14`, sem nenhuma opção explícita.

Foi utilizada a chamada de sistema `getrusage (RUSAGE_SELF)` antes e depois de cada cadeia de comandos a ser medida. Esta função fornece tempo de usuário e do sistema no momento de sua chamada. As cadeias de comandos são executadas dentro de laços com um número de iterações igual a um milhão.

O programa executa de uma só vez, apresentando os tempos parciais de execução das seguintes cadeias de comandos: um laço vazio, um laço para cada uma das operações de soma, subtração, multiplicação e divisão de números inteiros e ponto flutuante, um laço para cada operação de conversão de um número inteiro para um número ponto flutuante e vice-versa, um laço para chamada de uma função vazia, um laço para cada uma das chamadas de função seno, logaritmo, raiz quadrada e comparação de strings, um laço para execução de uma instrução de controle de fluxo (`if (i<0) {operando = i;} else {operando = i;}`), um laço para ler um inteiro de um arquivo, e um laço para escrever um inteiro em um arquivo.

Os tempos de cada cadeia são calculados pela diferença dos tempos medidos antes e

depois da cadeia de comandos executada, seguida da subtração do tempo do laço nulo e seguida ainda da divisão pelo número de iterações, obtendo-se assim o tempo individual do comando ou cadeia de comandos a ser medida. Ao final da execução, os tempos parciais são totalizados e apresentados.

3.4.1 Comparação entre Panalyser e Atsar

O `atsar` fornece uma ampla gama de informações sobre a utilização dos recursos do SO, conforme discutido na Seção 2.4.2. Estas informações apresentam, de forma agregada, a utilização total dos recursos do SO por todos os processos ativos no momento da medição. Com este programa não é possível determinar qual processo está fazendo maior utilização de qual recurso. Por exemplo, caso existam dois processos competindo pelo tempo do processador, as informações fornecidas mostrarão somente que a CPU está 0% *idle*, mas não é possível de se determinar qual a utilização da CPU por processo. Outro exemplo é a competição de processos pelo acesso ao disco, quando existe um grande número processos simultaneamente requisitando a carga de arquivos do disco para a memória. Apenas com a visão geral do volume de dados escritos e lidos não é possível determinar quais processos estão utilizando mais o disco para um determinado fim. Como as informações do `atsar` não são específicas a um processo, isto o torna pouco prático como ferramenta de aperfeiçoamento/depuração de programas. As informações fornecidas pelo **`atsar`** podem servir para apontar quais recursos do SO estão apresentando pontos de contenção. Se o sistema em questão não está sendo utilizado para um fim específico, ou seja, se o sistema não atende apenas um tipo de serviço principal, não é possível determinar quais processos estão causando contenção elevada em quais recursos.

A Tabela 3.2 compara a utilização de CPU pelo programa de teste nos domínios do sistema e do usuário medidas pelo **`panalyser`**, com a medida pelo **`atsar`** de todos os processos do sistema, inclusive do programa de teste. A grande discrepância entre as medidas de utilização de CPU no domínio do sistema para as duas ferramentas se deve ao fato do **`panalyser`** medir exclusivamente a utilização de recursos do programa de teste, enquanto o **`atsar`** inclui o tempo de tratamento de interrupções do processador em suas

medições, além das atividades dos demais processos, não relacionados ao programa de teste, executando no sistema [14].

Ferramentas	Ut. CPU usuário	Ut. CPU sistema
panalyser	10,59	33,03
atsar	22,72	51,74

Tabela 3.2: Comparação entre a utilização de CPU nos domínios de usuário e sistema medidos pelo panalyser e atsar.

3.4.2 Comparação entre o Panalyser e Strace

O strace registra todas as chamadas de sistema efetuadas por um processo, com os parâmetros e valor de retorno, bem como os sinais recebidos por um processo. Tais informações são indispensáveis para depuração de programas, porém, torna-se bastante complexo mensurar a utilização de recursos e fazer a otimização do sistema considerando apenas estas informações, pois não é possível quantificar a utilização de recursos, como CPU, memória e disco.

A Tabela 3.3 contém uma comparação entre os dados fornecidos pelo panalyser e strace, com a contabilização dos tipos de chamadas de sistema efetuadas pelo programa de teste. Em decorrência do comportamento determinístico do programa de teste, o número de chamadas de sistema foi igual em todas as repetições do experimento tanto medido pelo panalyser como pelo strace.

Chamadas de Sistema	Número Total	
	panalyser	strace
read	1082489	1082489
_llseek	2042233	2042233
write	1000248	1000248
open	6	6
getrusage	42	42
_mmap	10	10
munmap	4	4
close	5	5
fstat64	6	6
mprotect	2	2
brk	4	4
uname	1	1

Tabela 3.3: Número de chamadas de sistema efetuadas pelo programa de teste contabilizadas e classificadas pelo panalyser e strace.

3.4.3 Vantagens do Panalyser

O `panalyser` oferece tanto as informações fornecidas pelo `atsar` quanto pelo `strace`. As informações da estrutura `getrusage` fornecem a utilização de recursos como CPU, memória primária e memória secundária, enquanto a chamada de sistema `ptrace` permite o rastreamento, classificação e totalização das chamadas de sistema para um dado processo e seus processos filhos. A integração de ambos os tipos de informações na mesma medição facilita o entendimento das informações sobre a utilização de recursos pelos processos. Como exemplo é possível determinar rapidamente quais chamadas de sistema, efetuadas por um processo, estão causando o maior tempo de utilização do processador no domínio de sistema, ou ainda como o número de chamadas de sistema, efetuadas por um conjunto de processos do mesmo tipo, pode indicar a saturação de um recurso por estes processos.

3.5 Emprego das Medidas de Utilização de Recursos do Sistema Operacional

Durante o desenvolvimento de uma aplicação é importante levantar seu perfil de execução porque informações como o tempo utilizado em cada função e o número de chamadas a cada função permitem a verificação do comportamento esperado de um programa. Para esta finalidade existem ferramentas como o `gprof`. Entretanto estas informações não fornecem uma idéia precisa da utilização de chamadas de sistema pelo programa analisado, sendo esta informação de suma importância para se compreender a utilização de recursos pelo programa sob análise. Por exemplo, quando um programa executa muitas chamadas de sistema `read`, para ler dados de um arquivo, pode-se estudar a possibilidade de concentrar os dados a serem lidos, de forma a executar o menor número possível de chamadas de sistema `read`, uma vez que esta possui um alto custo de execução. Este custo se deve ao fato do processo em questão interromper sua execução e aguardar o retorno do sistema de arquivos indicando se a leitura obteve sucesso ou não. Enquanto isso, o escalonador de processos poderá escolher outro processo para executar. Como a freqüência com que o processo no exemplo efetua chamadas `read` é grande, isto acarretará o desperdício do

tempo de CPU em atividades de E/S que talvez sejam desnecessárias.

Para detectar pontos de contenção em um sistema, além do diagnóstico do ponto de contenção em si, é necessário desvendar as causas da contenção, ou seja, quais recursos do SO estão tendo sua capacidade utilizável esgotada por quais processos. Isto pode ser feito através da individualização dos dados de utilização de recursos do SO, por processo ou por tipo de processo. As ferramentas para medição de utilização de recursos que são distribuídas com os SOs geralmente fornecem dados sobre o sistema como um todo, sem discriminar os dados de utilização de recursos de cada processo. Além disso, tais ferramentas possuem um alto custo de execução e isto pode interferir significativamente nas medições efetuadas.

Em um sistema servidor, a utilização de recursos do SO e o comportamento dos processos que executam os serviços são motivo de acompanhamento constante por parte dos administradores do sistema, no esforço para obtenção do melhor desempenho. Os dados de utilização, neste caso, servem como base para obter melhor aproveitamento dos recursos que estão sendo exigidos pelos diferentes processos. Por exemplo, para determinar qual recurso deve ser melhorado para se obter maior vazão em um servidor Web, no qual a saturação na curva de vazão foi atingida com um número determinado de processos.

Dados como a diminuição do número de chamadas de sistema, a despeito do aumento do número de processos, indicam que a velocidade do processador está sendo insuficiente para que o número de tarefas executadas pelos processos, em seus *quanta*, seja o mínimo aceitável para compensar o número de trocas de contexto e proporcionar um aumento de vazão correspondente ao aumento do número de processos. Um aumento na capacidade do processador pode aumentar a vazão, desde que não se esbarre em outros fatores como a capacidade da rede e das memórias primária e secundária.

As informações de utilização de recursos do SO pelos processos podem servir igualmente para a caracterização de carga de trabalho para um sistema. Utilizando o exemplo do parágrafo anterior, os dados de utilização de CPU, memória primária e secundária juntamente com o perfil de chamadas de sistema executadas, servem para se determinar o impacto de uma determinada carga em um dos serviços do sistema. Por exemplo, para

duas cargas π e σ aplicadas a dois serviços Π e Σ do sistema, o recurso mais exigido pela operação concorrente dos servidores de Π e Σ é o sistema de arquivos. Sendo este compartilhado num único computador, a informação quanto a sobrecarga no sistema de arquivos poderia determinar a separação dos serviços em computadores diferentes para o melhoramento global da produção do sistema.

A partir da caracterização do comportamento de um sistema, para uma determinada carga e através dos dados de utilização de recursos, é possível fornecer parâmetros para a construção de um modelo do sistema, bem como validar um modelo através da variação dos parâmetros de entrada do modelo e a correspondente medição do sistema utilizando parâmetros correspondentes na carga de trabalho do sistema.

CAPÍTULO 4

ESTUDO DE CASO

Este capítulo contém um exemplo detalhado de uso do `panalyser` para a medição do servidor Web Apache, em condições extremas de funcionamento num ambiente controlado. As medições permitem demonstrar a utilidade, precisão e integração das informações de utilização de recursos, pelo processo principal do Apache e seus filhos, fornecidas pelo `panalyser`. Primeiramente é descrito o gerador de carga de trabalho utilizado para exercitar o Apache. A seguir é descrito o ambiente experimental e as medições efetuadas são então apresentadas e discutidas.

4.1 O Gerador de Carga SURGE

O gerador de carga de trabalho SURGE foi empregado para provocar a saturação do Apache no ambiente de testes que é descrito na Seção 4.2. O SURGE é um gerador de carga de trabalho que usa a abordagem analítica para gerar a carga de requisições HTTP [3], que depende de modelagem matemática para as várias características da carga de trabalho. O objetivo do SURGE é imitar o mais fielmente possível um fluxo de requisições HTTP originadas por uma população fixa de usuários Web. Os modelos usados para geração de carga adotados pelo SURGE podem ser examinados e variados para explorar diferentes aspectos da carga de trabalho.

O SURGE incorpora a idéia de número de equivalentes de usuário (EU) como uma medida da intensidade da carga de trabalho, em adição a diferentes tipos de distribuição para as seis características principais da carga de trabalho: tamanho dos arquivos, popularidade dos arquivos, localidade temporal, tamanho das requisições, intervalo entre requisições e referências a arquivos que compõem objetos em páginas HTTP.

A carga gerada pelo SURGE mantém um número maior de conexões abertas com o servidor Web, quando comparado ao SPECweb96, o que resulta numa alta utilização de

CPU. Em cargas altas, o SURGE gera tráfego de rede de alta variabilidade. Quando comparado com o SPECweb96, demonstrou gerar uma carga de trabalho mais realista e mais pesada para o sistema avaliado [3].

A versão do SURGE utilizada neste trabalho foi a 1.00a. O pacote SURGE é dividido em três partes: configurador do cliente, configurador do servidor, e gerador de carga ou de requisições. O configurador do cliente é responsável pela configuração das requisições que são feitas ao servidor Web, obedecendo as distribuições escolhidas para cada um dos 6 parâmetros que caracterizam uma carga de trabalho Web. O configurador do servidor gera os arquivos que comporão a carga do servidor Web, e o gerador de carga gera as requisições conforme configurado. O gerador de carga é dividido em um processo chamado `SURGEMaster` que é responsável por estabelecer a comunicação para sincronização dos processos cliente, e estes, por sua vez são os responsáveis pela geração das *threads*, que fazem o papel dos equivalentes de usuário e efetivamente geram as requisições ao processo servidor Web.

4.2 Ambiente Experimental

Os experimentos aqui relatados foram conduzidos em um ambiente composto de três PCs conectados através de uma rede local *Ethernet* de 100Mbps isolada de tráfego externo. O sistema servidor estava configurado com CPU AMD Athlon(tm) de 1311MHz com 512MB de RAM. Os sistemas cliente estavam configurados com CPU Intel Pentium III de 800MHz com 192MB de RAM. Os sistemas estavam executando Debian/GNU Linux Woody com *kernel* versão 2.4.16 otimizado para os respectivos processadores. O sistema servidor estava executando o servidor Apache 1.3.22 Debian/GNU. Os sistemas cliente estavam executando 6 processos cliente SURGE cada um.

O servidor Apache foi escolhido por ser uma aplicação que provê um serviço implementado sobre a camada de aplicação de redes TCP/IP, gerar processos filhos para atender conexões distintas, possuir *benchmarks*, ser altamente configurável e estar disponível para diversos SOs [3, 20]. Os parâmetros de configuração do Apache utilizados no experimento estão relacionados na Tabela 4.1. Dentre ele destacam-se o número máximo de clientes

que podem se conectar simultaneamente (`MaxClients`) e o número máximo de requisições que cada processo filho pode atender (`MaxRequestsPerChild`), que permitiram a saturação do sistema. Maiores detalhes sobre os parâmetros do Apache podem ser encontrados em [20].

Parâmetros	Valores
`Timeout`	300
`KeepAlive`	On
`MaxKeepAliveRequests`	150
`KeepAliveTimeout`	15
`MinSpareServers`	5
`MaxSpareServers`	10
`StartServers`	5
`MaxClients`	4800
`MaxRequestsPerChild`	10000000

Tabela 4.1: Parâmetros de configuração do Apache.

O SURGE apresenta a vazão média no sistema servidor em termos de requisições HTTP iniciadas por segundo e a latência média para o finalização das requisição HTTP nos clientes.

A versão do protocolo HTTP usada pelo SURGE no experimento é a 1.1, que possui como principais características as conexões persistentes, *pipelining*, compressão de documento no nível de *link*, e *caching* de documentos. Conexões persistentes reduzem o tráfego de pacotes, uma vez que apenas uma conexão TCP é estabelecida e utilizada durante uma sessão cliente de navegação em um servidor e isto significa que um maior número de conexões podem ser abertas no servidor a qualquer momento. Em [18], Mogul afirma que o gerenciamento de muitas conexões abertas no servidor pode aumentar a necessidade de processamento e carga em memória. O *pipelining* tem o efeito de reduzir o número de pacotes de requisições e respostas pelo melhor aproveitamento dos pacotes de dados.

4.3 Resultados dos Experimentos

4.3.1 Vazão Máxima e Saturação

O primeiro conjunto de experimentos efetuados mediu a vazão máxima do servidor Apache no ambiente experimental descrito na Seção 4.2, e serviu para demonstrar a saturação além da vazão máxima utilizável, pelo grande aumento da latência no atendimento das conexões e da diminuição da vazão. O número de EUs foi variado de 120 até 2400 em incrementos de 120 EU, divididos em dois sistemas-cliente e as cargas mais altas levaram o servidor a um estado de sobrecarga. Cada experimento foi executado durante 10 minutos, tempo suficiente para estabilizar medidas subseqüentes, conforme [3]. O SURGE foi configurado com 10000 arquivos únicos, resultando em aproximadamente 196 MBytes de dados de páginas HTTP no servidor. O experimento foi repetido 3 vezes e o conjunto de dados escolhido foi o que apresentou menor desvio padrão com relação a média dos dados das 3 repetições.

Os gráficos da Figura 4.1 mostram que acima de 2040 equivalentes de usuário o servidor chega a um estado saturado, no qual a vazão média de requisições por segundo no servidor começa a declinar e a latência média para o finalização das requisições nos clientes tem seus valores sensivelmente aumentados, quando comparada aos valores da latência em operação não saturada.

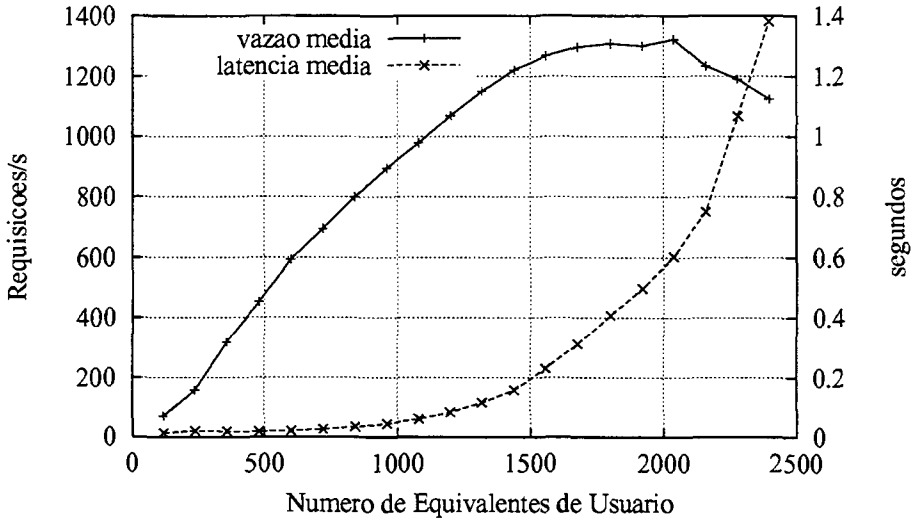

Figura 4.1: Vazão dos processos Apache no servidor e latência média para a finalização das requisições nos clientes.

4.3.2 Vazão x Latência

O segundo conjunto de experimentos repete as medições do primeiro conjunto de experimento em 5 pontos das curvas de vazão e latência. Para isto foram escolhidos os pontos com 840 e 1560 equivalentes de usuário, abaixo do ponto de inflexão da curva de vazão; 1800 e 2040 equivalentes de usuário, onde ocorre vazão máxima; e 2280 equivalentes de usuário, onde ocorre a saturação. Estes pontos foram escolhidos de forma a reproduzir os pontos críticos nas curvas do experimento anterior, desde antes do ponto de inflexão da curva de vazão até a saturação. As curvas de vazão e latência para os pontos escolhidos são mostradas na Figura 4.2. Para que o servidor estivesse nas mesmas condições do experimento anterior, ou seja, com os *caches* do sistema de arquivos pré-aquecidos, para cada um dos pontos de medição foi feito o exercício do servidor com número de equivalentes de usuário imediatamente inferior ao do ponto medido (como empregado no primeiro conjunto de experimentos) seguido pela medição com o número de usuários escolhido. O experimento foi repetido 5 vezes para cada ponto escolhido e o conjunto de dados apresentado foi o que apresentou menor desvio padrão em relação a média dos dados das 5 repetições, em cada ponto.

O intervalo de amostragem usado nestes experimentos foi de 5 segundos, para que fosse possível comparar alguns dos resultados obtidos com dados do `atsar` conforme mostrado na Seção 3.4.1. O `atsar` provocou um grande impacto no sistema com um intervalo de amostragem menor do que 5 segundos e portanto a comparação de dados não seria válida, pois o impacto causado pelo `panalyser` é menor. O intervalo de amostragem de processos filhos foi de um rastreado a cada 20 processos criados pelo Apache. A escolha deste intervalo de amostragem foi motivada pelo fato da configuração do SURGE, empregada nos experimentos, utilizar a versão 1.1 do protocolo HTTP. Como esta versão do protocolo mantém conexões persistentes, o tempo de duração dos processos atendendo às requisições é longo o suficiente para resultar na existência de um grande número de processos do Apache simultaneamente em memória. Caso todos os processos filhos fossem rastreados o próprio `panalyser` contribuiria para a sobrecarga do sistema.

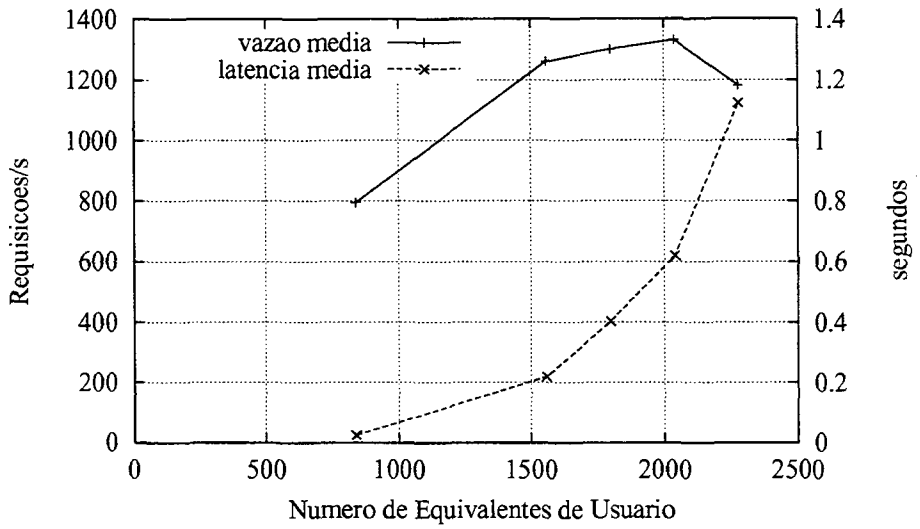

Figura 4.2: Vazão dos processos Apache no servidor medida pelo `panalyser` e latência média para a finalização das requisições nos clientes.

A Tabela 4.2 fornece a variação percentual na vazão e latência do Apache para a média dos dados de vazão e latência das 5 repetições do experimento nos 5 pontos escolhidos, informadas pelo SURGE, com e sem a monitoração do `panalyser`. As variações percentuais para a vazão e latência foram de menos de 2% demonstrando a baixa distorção nas medições causada pelo `panalyser`.

Equivalentes de Usuário	Com `panalyser`		Sem `panalyser`		Variação	
	vazão média (req/s)	latência média (s)	vazão média (req/s)	latência média (s)	vazão (%)	latência (%)
840	793,23	0,024	798,25	0,024	0,63	0,00
1560	1256,17	0,222	1265,94	0,221	0,77	0,46
1800	1292,39	0,403	1306,87	0,399	1,11	0,99
2040	1298,71	0,648	1321,33	0,644	1,71	0,62
2280	1181,67	1,078	1189,83	1,065	0,69	1,21

Tabela 4.2: Variação percentual gerada pelo `panalyser` na vazão e latência médias do Apache.

A Tabela 4.3 demonstra o comportamento probabilístico do Apache submetido ao SURGE, somado às pequenas flutuações na execução de processos nos sistemas cliente e servidor, não relacionados ao Apache, executando no sistema. A variação entre o menor e maior valor das repetições do experimento, nas mesmas condições (para 1560 equivalentes de usuário) são de menos de 1% para a vazão e menos de 4% para a latência. Para

2280 equivalentes de usuário a variação é de menos de 13% para vazão e menos de 17% para latência. Com 2280 equivalentes de usuário a maior variação percentual é explicada pela saturação do sistema causada pelo grande número de processos Apache com picos de atividade ocorrendo em momentos diferentes e pelo comportamento probabilístico do experimento em cada repetição, não permitindo a operação eficiente do escalonador de processos e uso adequado da memória secundária, conforme detalhado a seguir.

Repetição do Experimento	1560 EU		2280 EU	
	vazão média (req/s)	latência média (s)	vazão média (req/s)	latência média (s)
1	1259,54	0,225	1203,31	1,072
2	1258,49	0,227	1277,42	0,883
3	1250,72	0,220	1180,50	1,122
4	1252,00	0,221	1123,56	1,062
5	1260,08	0,218	1123,56	1,254

Tabela 4.3: Vazão e latência do Apache medido pelo `panalyser` em 5 repetições do experimento.

4.3.3 Utilização do Processador

Nos gráficos da Figura 4.3 e na Tabela 4.4 são apresentadas as médias de utilização de CPU nos domínios de usuário e sistema. Estes valores foram obtidos através da média dos tempos de utilização de CPU nos respectivos domínios em intervalos de 5 segundos. Os dados apontam uma maior utilização da CPU no domínio do sistema nos cinco pontos escolhidos nas curvas de vazão e latência, mostradas nos gráficos da Figura 4.2. Comportamento semelhante foi detectado nos experimentos descritos em [7, 11], indicando que o tempo despendido no processamento de chamadas de sistema é maior que o tempo despendido no processamento de URLs e logs. Observa-se que na saturação, a utilização de CPU no domínio de usuário é menor do que nos outros quatro pontos anteriores à saturação. Já para o domínio de sistema ocorre o oposto, devido ao tempo de espera dos processos pelas respostas das chamada de sistema efetuadas, e ao grande número de processos e o conseqüente alongamento da fila de processos prontos para executar.

Equivalentes de Usuário	% de CPU Usuário	% de CPU Sistema	*Page faults*/s	Chamadas de Sistema			
				reads/s	*writes*/s	*forks*/s	*socketcalls*/s
840	3,35	6,13	467,67	176,20	246,03	0,51	59,08
1560	4,22	8,61	500,36	152,63	200,31	0,88	55,81
1800	4,94	12,64	662,79	201,54	265,20	0,96	75,31
2040	3,34	8,36	449,12	149,92	187,24	0,97	61,29
2280	2,16	19,20	275,77	92,44	116,70	0,87	51,59

Tabela 4.4: Médias das medições dos recursos utilizados pelo Apache.

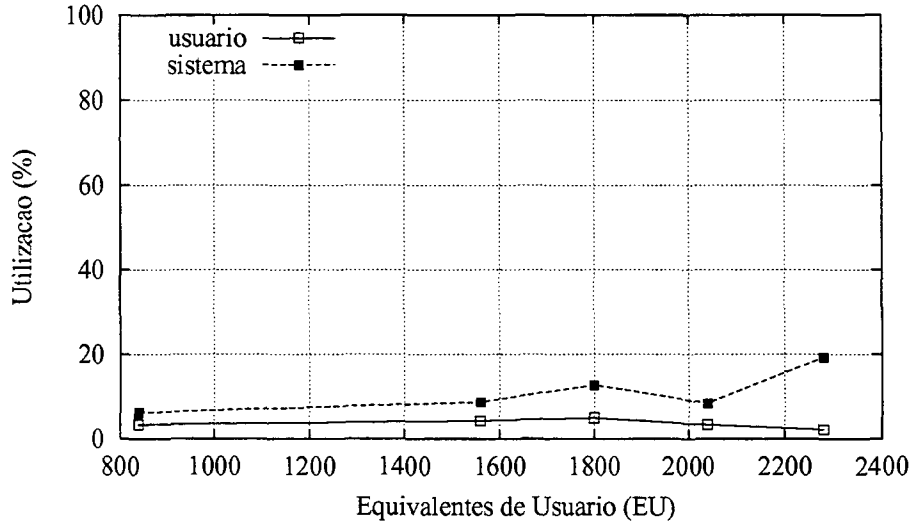

Figura 4.3: Média de utilização de CPU para 840 até 2280 EU, nos domínios do usuário e sistema.

4.3.4 Page Faults

No gráfico da Figura 4.4 é apresentado o número de *page faults* que os processos incorreram nos cinco pontos medidos, ou seja, o número de faltas que exigiram a carga de páginas do disco para a memória. A média de *page faults* que os processos incorreram é apresentada na Tabela 4.4. O menor número de *page faults* ocorrido no ponto de saturação é explicado pelo maior número de processos do mesmo tipo em memória e conseqüentemente a maior probabilidade de a página requerida estar em memória, um efeito similar a uma *cache* com o código dos programas. Como cada processo ficará aguardando até que as páginas requisitadas sejam carregadas na memória principal a partir da memória secundária, esta operação é de alto custo, e portanto de relevância para avaliação de desempenho de um processo [2].

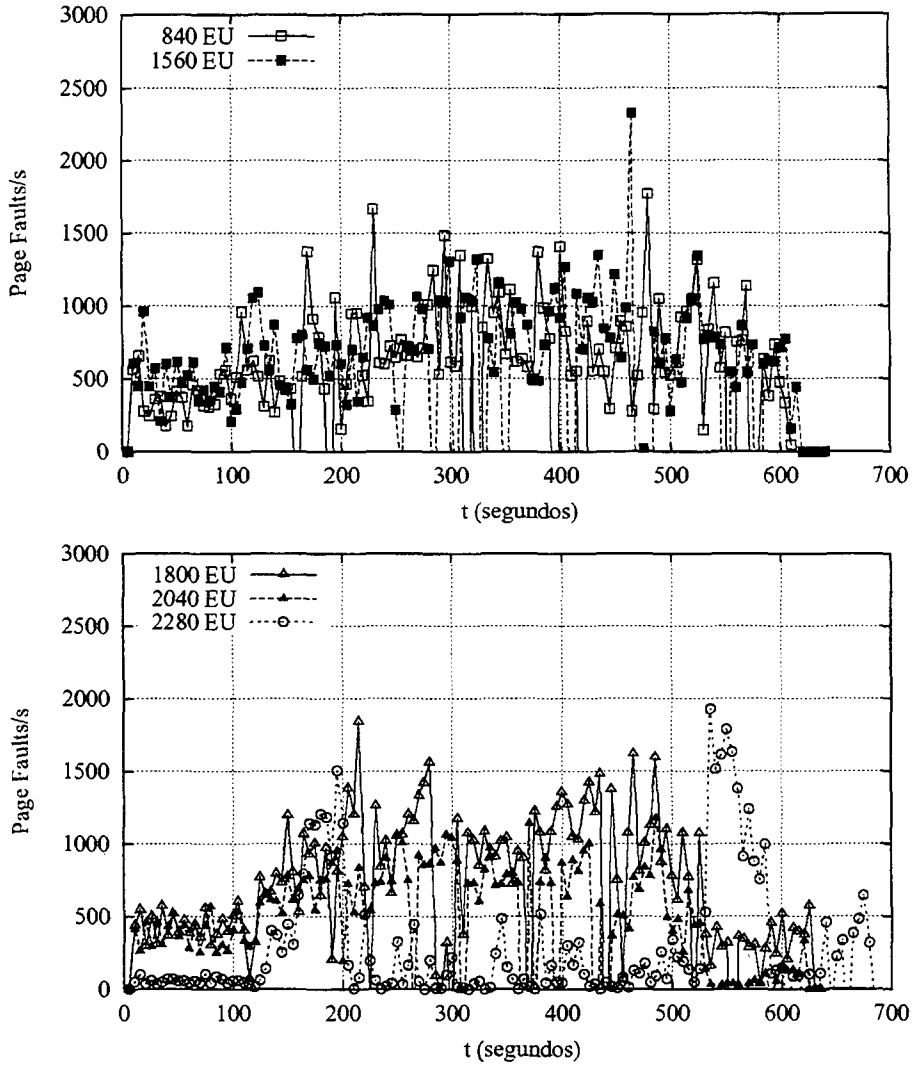

Figura 4.4: Número de *page faults*/s para 840 e 1560 EU (topo), e 1800 até 2280 EU (base).

4.3.5 Chamadas de Sistema

Nos gráficos das Figuras 4.5 , 4.6 , 4.7 e 4.8 são mostradas as chamadas de sistema `read`, `write`, `fork` e `socketcall`, efetuadas pelos processos do Apache durante execução nos cinco pontos medidos. A média das chamadas de sistema de cada tipo é apresentada na Tabela 4.4. É notável a correspondência entre o número de ocorrências de cada tipo de chamada de sistema em cada um dos pontos das curvas de vazão e latência. Na saturação observa-se um menor número de chamadas de sistemas de cada tipo, durante a maior parte da medição. Isto decorre do grande número de processos aguardando a sua vez de ser executado, reduzindo significativamente a quantidade de tempo do processador

dedicada a cada processo. Como a capacidade do processador, vazão da memória e disco para uma dada latência são limitados fisicamente, quanto menor o tempo de processador disponível a cada processo, menor a quantidade de tarefas executadas pelos processos. O número de trocas de contexto necessárias, proporcional ao número de processos, contribui para o estado de saturação, pois diminui ainda mais o tempo de processamento, acessos à memória e ao disco possíveis a cada processo.

No ponto de saturação, o grande número de processos atendendo a requisições faz com que os recursos disponíveis a cada processo sejam muito reduzidos, acarretando assim a diminuição da vazão, aumento da latência e a redução do número de chamadas de sistemas dos processos.

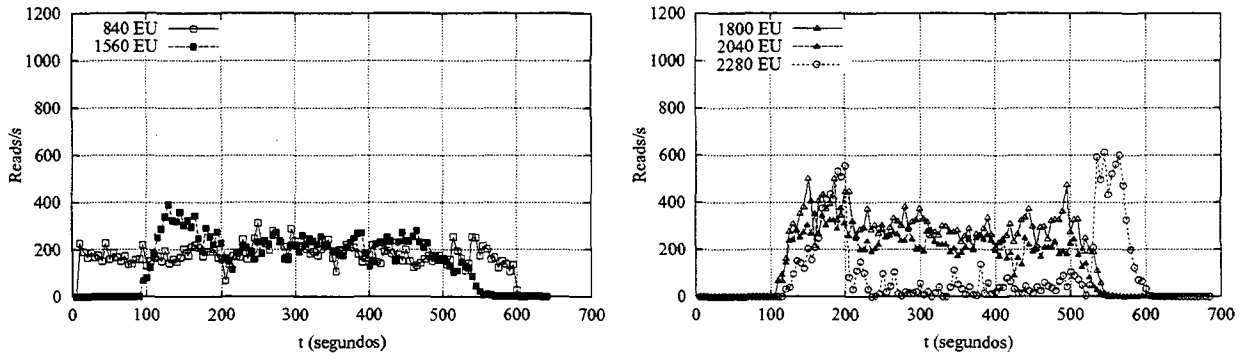

Figura 4.5: Número de *reads*/s para 840 e 1560 EU (esquerda), e 1800 até 2280 EU (direita).

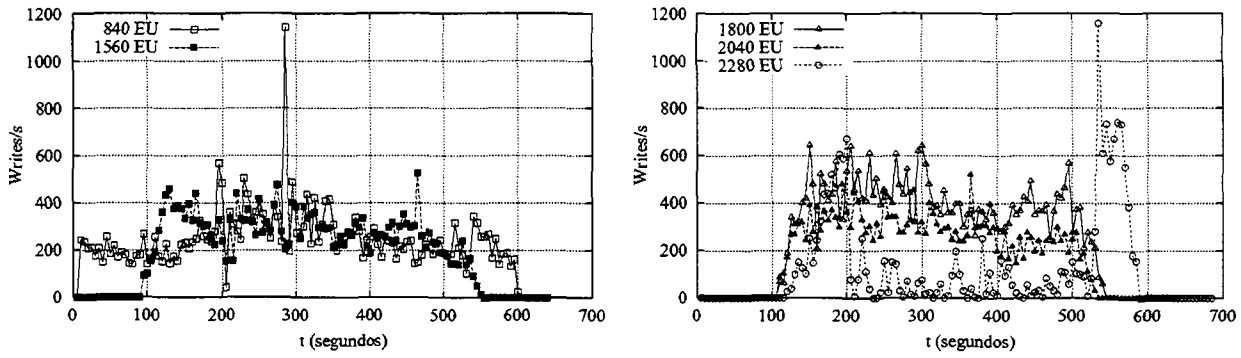

Figura 4.6: Número de *writes*/s para 840 e 1560 EU (esquerda), e 1800 até 2280 EU (direita).

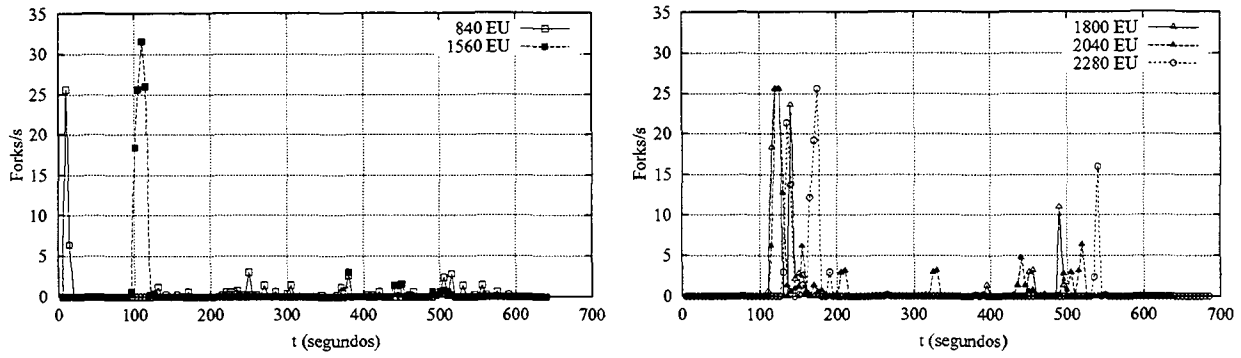

Figura 4.7: Número de *forks*/s para 840 e 1560 EU (esquerda), e 1800 até 2280 EU (direita).

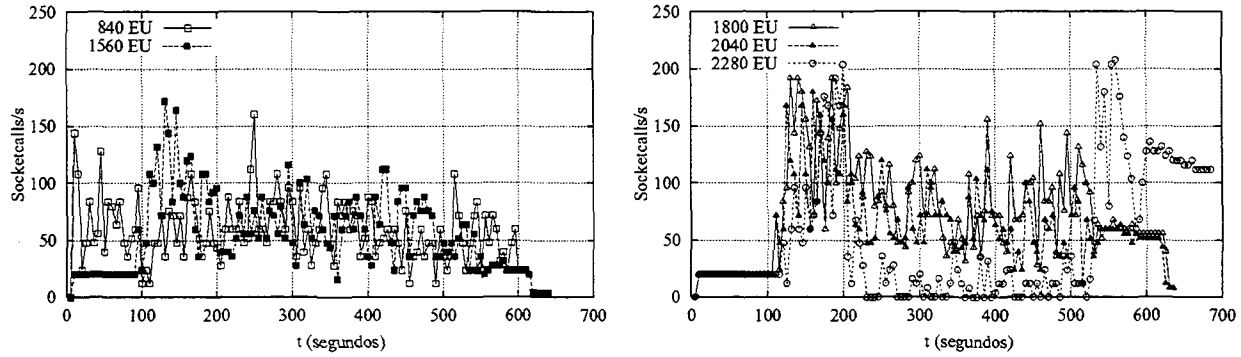

Figura 4.8: Número de *socketcalls*/s para 840 e 1560 EU (esquerda), e 1800 até 2280 EU (direita).

Neste estudo de caso foram observadas apenas 4 tipos de chamadas de sistema efetuadas pelos processos do Apache, para caracterizar o tipo de informação fornecida pelo `panalyser` sobre as chamadas de sistema. As demais chamadas de sistema efetuadas não foram apresentadas porque a análise completa sobre todas as chamadas de sistema efetuadas pelo Apache não faz parte do escopo deste trabalho. As chamadas desconsideradas foram: `kill`, `wait4`, `exit`, `open`, `close`, `time`, `getpid`, `times`, `brk`, `gettimeofday`, `mmap`, `munmap`, `socketcall`, `sigreturn`, `uname`, `_llseek`, `_newselect`, `writev`, `rt_sigaction`, `rt_sigprocmask`, `mmap2`, `stat64`, `lstat64`, `fstat64`, `geteuid32`, `setgroups32`, `setuid32`, `setgid32` e `fcntl64`. Estas correspondem a pouca atividade do servidor Apache no domínio do sistema.

Nos gráficos da Figura 4.6 aproximadamente aos 280 segundos no gráfico da esquerda e aproximadamente aos 530 segundos no gráfico da direita, observam-se pontos com valores

muito superiores ao restante das medições. A ocorrência de tais valores é explicada pelo fato de o intervalo entre amostragens não ser exatamente aquele declarado no parâmetro de chamada do `panalyser`. Estas diferenças na duração do intervalo decorrem do tempo de processamento do `panalyser`. Este tempo normalmente é muito pequeno e pode ser desconsiderado, mas pode ser alongado, por exemplo, por uma troca de contexto efetuada no exato momento em que a amostragem está sendo processada.

É importante observar que o número de `writes/s` e `reads/s` não implicam necessariamente em acesso a disco, uma vez que no primeiro caso o SO se utiliza de escrita preguiçosa, ou seja, acumula os dados em memória antes de serem escritos no disco, só o fazendo quando estes dados são requisitados, ou quando o espaço em memória destinado a esta função é esgotado. No caso da chamada de sistema `read` o que ocorre é a leitura a partir do disco apenas quando os dados requisitados não estão em memória.

4.3.6 Conclusão Sobre a Saturação do Apache

A saturação do servidor é constatada pelo declínio da vazão média de requisições por segundo no servidor, acompanhado de um sensível aumento da latência média para a finalização das requisições nos clientes quando comparada aos valores da latência em operação não saturada.

O sistema foi levado à saturação ao empregar-se um número de processos superior à capacidade de gerência eficiente pelo escalonador do SO da fila de processos prontos para execução, dadas as limitações de velocidade e capacidade do processador e memória do sistema utilizado no experimento. Este comportamento foi observado na maior utilização de CPU no domínio do sistema durante a saturação, juntamente com uma diminuição do número de chamadas de sistema executadas pelos processos do Apache. Daí conclui-se que a utilização maior de CPU no domínio do sistema durante a saturação se deve ao escalonador de processos e não à execução das chamadas de sistema pelos processos do Apache. A diminuição do número de *page faults* contribui positivamente para a redução da carga do sistema, mas não significativamente perante as afirmações do parágrafo anterior.

CAPÍTULO 5

CONCLUSÃO E TRABALHOS FUTUROS

O Sistema Operacional torna disponíveis e gerencia os recursos de *hardware* de um sistema de computadores para os processos. Os dados de utilização destes recursos pelos processos são de grande relevância para o desenvolvimento e avaliação de desempenho de aplicações, o diagnóstico de pontos de contenção, a caracterização de carga, e o levantamento de parâmetros e validação de modelos de um sistema.

Este trabalho apresenta a ferramenta `panalyser` que foi projetada e implementada para medição de utilização de recursos pelos processos no SO GNU/Linux. O `panalyser` é um monitor que opera em batelada sendo controlado por eventos e amostragem e fornece dados sobre a utilização de CPU, memória primária e secundária, e classificação e totalização das chamadas de sistema dos processos monitorados. Os processos monitorados podem ser quaisquer processos em execução inclusive processos filhos de processos sob exame.

O `panalyser` causa baixa distorção nas medidas porque baseia-se nas chamadas de sistema `ptrace`, `wait4` e `getrusage` para fazer a amostragem de utilização de recursos do SO e rastreamento das chamadas de sistema do processo analisado. Ferramentas como `atsar`, `ps`, `top`, utilizam-se da leitura freqüente do pseudo sistema de arquivos `/proc`, cuja leitura têm um maior custo de processamento do que as chamadas de sistema utilizadas pelo `panalyser`. Além disso o `panalyser` é portável para todas as plataformas de *hardware* suportadas pelo Linux.

A informação precisa sobre o número de chamadas de sistema invocadas por um programa é de vital importância para se compreender a forma e a intensidade da utilização de recursos do SO pelo programa. Para desvendar as causas dos pontos de contenção em um sistema, ou seja, quais recursos do SO estão tendo sua capacidade utilizável esgotada por quais processos, é necessário individualizar os dados de utilização de recursos por pro-

cesso, ou por tipo de processo. Os dados de utilização de recursos fornecem parâmetros para o ajuste na quantidade e qualidade dos recursos do sistema que são mais exigidos no provimento dos serviços de um sistema. Estes dados também permitem identificar o impacto de cargas de trabalho sobre os servidores. As informações de utilização de recursos servem ainda para fornecer parâmetros e validar modelos.

O preenchimento incompleto da estrutura `rusage` pelo *kernel* do Linux reduz a gama de informações fornecidas pelo `panalyser`, mas não impede sua utilização para atingir os objetivos de uso propostos. A implementação do `panalyser` contempla os campos não preenchidos da estrutura, de forma tal que quando seu preenchimento for implementado no *kernel*, estas informações serão facilmente disponibilizadas pelo `panalyser`.

A comparação dos dados fornecidos pelo `panalyser`, `atsar` e `strace` na medição de um programa de teste de comportamento determinístico atestaram a correção dos dados fornecidos pelo `panalyser`.

Um estudo de caso demonstra o uso do `panalyser` no monitoramento dos recursos utilizados por um servidor Apache, quando este é submetido a diferentes intensidades de cargas, até a saturação do sistema como um todo. As medições mostraram que o percentual de utilização de CPU no domínio do sistema é superior a do domínio do usuário, em concordância com o descrito na literatura [7, 11]. O sistema foi levado a saturação através do número de processos superior à capacidade de gerência eficiente pelo escalonador do SO da fila de processos prontos para execução, dadas as limitações de velocidade e capacidade do processador e memória do sistema utilizado no experimento. Este comportamento foi observado na maior utilização de CPU no domínio do sistema durante a saturação juntamente com uma diminuição do número de chamadas de sistema executadas pelos processos do Apache, de onde conclui-se que a utilização maior de CPU no domínio do sistema durante a saturação se deve ao escalonador de processos e não à execução das chamadas de sistema pelos processos do Apache.

Trabalhos Futuros Uma extensão deste trabalho seria a implementação do preenchimento dos campos vazios da estrutura `rusage` no *kernel* do Linux, permitindo assim a

ampliação da gama de informações fornecidas pelo `panalyser` para os usos mencionados anteriormente. O `panalyser` está sendo adequado às normas da *GNU General Public License* (GNU GPL) com a finalidade de disponibiliza-lo à comunidade Linux. Outro estudo de caso a ser efetuado é a avaliação e proposição de correções na distribuição de recursos de *hardware* entre aplicações executadas no núcleo de servidores do Laboratório do Departamento de Informática da UFPR.

APÊNDICE A

PROGRAMA PANALYSER

```
1  /*
    * Copyright (c) 2001-2002 Martin Alain Kretschek <martink@osite.com.br>
    * All rights reserved.
    *
    * Redistribution and use in source and binary forms, with or without
    * modification, are permitted provided that the following conditions
    * are met:
    * 1. Redistributions of source code must retain the above copyright
    *    notice, this list of conditions and the following disclaimer.
    * 2. Redistributions in binary form must reproduce the above copyright
    *    notice, this list of conditions and the following disclaimer in the
    *    documentation and/or other materials provided with the distribution.
    * 3. The name of the author may not be used to endorse or promote products
    *    derived from this software without specific prior written permission.
    *
    * THIS SOFTWARE IS PROVIDED BY THE AUTHOR ''AS IS'' AND ANY EXPRESS OR
    * IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
    * OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED.
    * IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT,
    * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
    * NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
    * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
    * THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
    * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF
    * THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
    *
    *      $Id: definitions.h,v 1.0Release 2002/01/10 11:30:00 martin Exp $
    */

   #include <stdio.h>
   #include <stddef.h>
   #include <stdlib.h>
   #include <sys/resource.h>
   #include <syscall.h>
   #include <sys/types.h>
   #include <sys/stat.h>
37 #include <sys/wait.h>
   #include <sys/ptrace.h>
   #include <asm/ptrace.h>
   #include <unistd.h>
   #include <errno.h>
   #include <signal.h>
   #include <sys/times.h>
   #include <time.h>
   #include <sys/time.h>
   #include <fcntl.h>
   #include <string.h>
   #include <sys/param.h>
   #include <pthread.h>

   /*
     Constants
   */
   /* Conversion from seconds to microseconds and vice versa */
   #define M 1000000

   /* Default measure time in seconds */
   #define DEFAULTTIME 60

   /* Default SIGSTOPs interval in seconds*/
   #define DEFAULTSLEEP 5
```

```
/* Current number of syscalls on Linux */
#define NO_SYS_CALLS_PLUSONE 222

/* Swap vector size */
#define SWAPVECTORSIZE 64

/* Number of digits of PID */
#define N_PROC_DIGIT 7

/* Maximum number of followforks */
#define MAXFORKSTOFOLLOW 20000
74
/* Number of tries mutex lock, waiting for follow forks threads to finish */
#define MUTEX_TRYL_TIMEOUT 3

#define TRUE 1
#define FALSE 0

/* flag values*/
#define STARTUP    00001 /* We have just begun ptracing this process */
#define READFSWAPL 00002 /* Read from swap later when follow forks */
#define INSYSCALL  00004 /* A system call is in progress */
#define ATTACHED   00010 /* Process is not our own child */
#define EXITING    00020 /* As far as we know, this process is exiting */
#define SUSPENDED  00040 /* Process has done a wait(4), that can
                            not be allowed to complete just now */
#define STATISTICS 00100 /* Enable statistics */
#define OPTFFORK   00200 /* Process should have forks followed */
#define FOLLOWFORK 00400
#define UNTILEND   01000 /* Trace process until it ends */
#define WAITEXECVE 02000 /* ignore SIGTRAP after excve */
#define SAVERESULT 04000 /* Save the results */

/* flag_swap values*/
#define FIRSTSWAP  00001 /* First swap to open swap file */
#define RESULTSWAP 00002 /* The swap was made */

/* ord_criterio values */
#define NSYSCALLS 00001 /* Number of syscalls */
#define UTIME     00002 /* User time */
#define STIME     00004 /* System time */

/*
  Data Structures
*/
typedef struct st_proc_stat element_proc_stat, *pointer_proc_stat;

struct st_proc_stat {
111   int starttime;
  unsigned int vsize;
  unsigned int rss;
  double cpu_percentage;
};

/* linked list */
typedef struct st_lista registro, *lista_db;

struct st_lista {
  unsigned long int indice; /* numero de insercao do no */
  double momento_insercao; /* momento de isercao do no em tempo real */
  struct rusage sys_res_usage; /* dados de consumo de recusros do no */
  element_proc_stat proc_stat; /* dados do /proc/PID/stat do no */
  unsigned long int sys_call_in_cont; /* numero de chamadas de sistema ate o momento */
  unsigned long int sys_call_no[NO_SYS_CALLS_PLUSONE]; /* classificacao das chamadas de sistema ate o momento */
  struct timeval diff_ru_utime; /* diferenca entre o tempo de usuario usado deste no com o anterior */
  struct timeval diff_ru_stime; /* diferenca entre o tempo de sistema usado deste no com o anterior */
  unsigned long int diff_sys_call_in_cont; /* diferenca entre o numero de chamadas de sistema neste no com o anterior
  lista_db prox; /* ponteiro para o proximo no */
  lista_db ant; /* ponteiro para o no anterior */
};

/* statistic rusage structure */
```

```
struct statistic_rusage {
  struct timeval ru_utime_st; /* user time used */
  struct timeval ru_stime_st; /* system time used */
  double ru_maxrss_st;          /* maximum resident set size */
  double ru_ixrss_st;       /* integral shared memory size */
  double ru_idrss_st;       /* integral unshared data size */
  double ru_isrss_st;       /* integral unshared stack size */
  double ru_minflt_st;          /* page reclaims */
  double ru_majflt_st;          /* page faults */
  double ru_nswap_st;       /* swaps */
  double ru_inblock_st;         /* block input operations */
  double ru_oublock_st;         /* block output operations */
  double ru_msgsnd_st;          /* messages sent */
148 double ru_msgrcv_st;         /* messages received */
  double ru_nsignals_st;        /* signals received */
  double ru_nvcsw_st;       /* voluntary context switches */
  double ru_nivcsw_st;          /* involuntary context switches */
};

typedef struct st_thread_analyser element_thread_analyser, *pointer_thread_analyser;

struct st_thread_analyser {
  pthread_cond_t cond_i;
  pthread_mutex_t mutex_i;
  unsigned int change_i;
  unsigned long int i;
  pthread_mutex_t mutex_sigstop_sent;
  unsigned int sigstop_sent;
  pthread_mutex_t mutex_analyser_thread_signalsender_error;
  unsigned int analyser_thread_signalsender_error;
  pid_t pid;
  unsigned short int flag;
  unsigned int seconds;
  unsigned long int useconds;
  unsigned long int tam_loop;
  int debug;
};

typedef struct st_thread_ffork element_thread_ffork, *pointer_thread_ffork;

struct st_thread_ffork {
  unsigned int ffeax_number;
  pid_t ffeaxvector[MAXFORKSTOFOLLOW];
  pid_t pid;
  unsigned short int flag;
  unsigned short int flag_swap;
  unsigned int seconds;
  unsigned long int useconds;
  unsigned long int tam_loop;
  FILE *outf;
185 char *outfname;
  FILE *stf_rusage;
  char *stfname_rusage;
  FILE *stf_syscall;
  char *stfname_syscall;
  char *swap_outfname;
  unsigned short int criterio_ord;
  time_t process_start_time;
  time_t panalyser_start_time;
  double real_run_time;
  struct timeval effective_run_time;
  pointer_thread_ffork ffanalyseproc[MAXFORKSTOFOLLOW];
  pthread_mutex_t mutex_ffanalyser_thread_end;
  unsigned int ffanalyser_thread_end;
  unsigned int interval_fforks;
  int debug;
};

typedef struct st_process_detach element_process_detach, *pointer_process_detach;

struct st_process_detach {
  pid_t pid;
  int sig;
```

```
};


/*
  Functions
*/
/* control, data storage and statistics functions */
int data_swap_in (registro [], pid_t, char *, unsigned int, unsigned short int, int);
int data_swap_out (lista_db *, pid_t, char *, int);
int init_list (lista_db *);
int insert_end (lista_db *, unsigned long int, double, struct rusage, element_proc_stat, unsigned long int, unsigned l
lista_db localizar_fim (lista_db);
int del_list (lista_db *);
int timeval_subtract (struct timeval *, struct timeval, struct timeval, int);
222 int timeval_add (struct timeval *, struct timeval, struct timeval, int);
int timeval_div (struct timeval *, struct timeval, double, int);
int timeval_pow (struct timeval *, struct timeval, double, int);
int timeval_sqrt (struct timeval *, struct timeval, int);
int mean_calc (lista_db, struct statistic_rusage *, double *, double *, double [], int);
int variance_calc (lista_db, struct statistic_rusage *, double *, double *, double [], struct statistic_rusage, double
int order_list (lista_db, lista_db *, unsigned int, unsigned short int, int);
double median_calc (lista_db, unsigned short int, int);
int calc_present (lista_db, unsigned short int, unsigned short int, FILE *, FILE *, FILE *, int);

/* analysis  functions */
void *analyser_thread_signalsender (void *);
int process_detach(element_process_detach);
int upeek(int, long int, long int *);
void *analyser (void *);

/* statistics and print functions */
int print_getrusage_summary (struct rusage, struct statistic_rusage, struct statistic_rusage, struct rusage, double, d
int print_proc_stat_summary (element_proc_stat, FILE *, int);
int print_sys_call_summary (long int [], char * [], double [], double [], long int [], FILE *, int);
int print_list (lista_db, unsigned short int, char * [], struct statistic_rusage, struct statistic_rusage, double, dou
int print_getrusage_summary_table (unsigned long int, double, struct rusage, struct statistic_rusage, struct statistic
int print_sys_call_summary_table (unsigned long int, double, long int [], char * [], double [], double [], long int []
int gen_proc_summary (struct timeval, struct timeval, struct timeval, int, double *, double *, struct timeval *, int);
int sys_call_name_init (char * []);
```

```
1   /*
     * Copyright (c) 2001-2002 Martin Alain Kretschek <martink@osite.com.br>
     * All rights reserved.
     *
     * Redistribution and use in source and binary forms, with or without
     * modification, are permitted provided that the following conditions
     * are met:
     * 1. Redistributions of source code must retain the above copyright
     *    notice, this list of conditions and the following disclaimer.
     * 2. Redistributions in binary form must reproduce the above copyright
     *    notice, this list of conditions and the following disclaimer in the
     *    documentation and/or other materials provided with the distribution.
     * 3. The name of the author may not be used to endorse or promote products
     *    derived from this software without specific prior written permission.
     *
     * THIS SOFTWARE IS PROVIDED BY THE AUTHOR ''AS IS'' AND ANY EXPRESS OR
     * IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
     * OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED.
     * IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT,
     * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
     * NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
     * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
     * THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
     * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF
     * THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
     *
     *      $Id: panalyser.c,v 1.0Release 2002/01/10 22:43:00 martin Exp $
     */

    #include "definitions.h"

    void
    *analyser_thread_signalsender (void * void_analyseproc)
    {
      pointer_thread_analyser analyseproc;
      int mutex_error;
37    int cond_error;
      int *status = NULL;
      struct timeval tempo_inicial_tv, tempo_final_tv, tempo_decorrido_tv;
      struct timeval before_sigstop_tv, after_sigstop_tv, between_sigstops_tv;

      analyseproc = void_analyseproc;

      if ((status = (int *) malloc (sizeof(int)))!=NULL){
        *status = 1;
      }else{
        if ((mutex_error = pthread_mutex_lock(&(analyseproc->mutex_analyser_thread_signalsender_error))) != 0){
          fprintf (stderr, "analyser_thread_signalsender - pthread_mutex_lock error: %s.\n", strerror (mutex_error));
        }
        analyseproc->analyser_thread_signalsender_error = TRUE;
        if ((mutex_error = pthread_mutex_unlock(&(analyseproc->mutex_analyser_thread_signalsender_error))) != 0){
          fprintf (stderr, "analyser_thread_signalsender - pthread_mutex_unlock error: %s.\n", strerror (mutex_error));
        }
        perror ("analyser_thread_signalsender error");
        return status;
      }

      pthread_testcancel();

      while (TRUE){

        if ((mutex_error = pthread_mutex_lock(&(analyseproc->mutex_i))) != 0){
          fprintf (stderr, "analyser_thread_signalsender - pthread_mutex_lock error: %s.\n", strerror (mutex_error));
          free (status);
          status = NULL;
          break;
        }

        if ((analyseproc->i > analyseproc->tam_loop) && !(analyseproc->flag & UNTILEND)){
          if ((mutex_error = pthread_mutex_unlock(&(analyseproc->mutex_i))) != 0){
            fprintf (stderr, "analyser_thread_signalsender - pthread_mutex_unlock error: %s.\n", strerror (mutex_error));
            free (status);
            status = NULL;
```

```
74          break;
          }
          break;
        }

        if ((mutex_error = pthread_mutex_unlock(&(analyseproc->mutex_i))) != 0){
          fprintf (stderr, "analyser_thread_signalsender - pthread_mutex_unlock error: %s.\n", strerror (mutex_error));
          free (status);
          status = NULL;
          break;
        }

        if (gettimeofday (&tempo_inicial_tv, NULL) < 0){
          perror ("analyser_thread_signalsender - gettimeofday error");
          free (status);
          status = NULL;
          break;
        }

        sleep (analyseproc->seconds);
        usleep (analyseproc->useconds);

        if (gettimeofday (&tempo_final_tv, NULL) < 0){
          perror ("analyser_thread_signalsender - gettimeofday error");
          free (status);
          status = NULL;
          break;
        }

        if (timeval_subtract (&tempo_decorrido_tv, tempo_final_tv, tempo_inicial_tv, analyseproc->debug) != 0){
          perror ("analyser_thread_signalsender - timeval_subtract error");
          free (status);
          status = NULL;
          break;
        }

        if (tempo_decorrido_tv.tv_sec < analyseproc->seconds){
111       sleep (analyseproc->seconds - tempo_decorrido_tv.tv_sec);
        }
        if (tempo_decorrido_tv.tv_usec < analyseproc->useconds){
          usleep (analyseproc->useconds - tempo_decorrido_tv.tv_usec);
        }

        pthread_testcancel();

        if ((mutex_error = pthread_mutex_lock(&(analyseproc->mutex_sigstop_sent))) != 0){
          fprintf (stderr, "analyser_thread_signalsender - pthread_mutex_lock error: %s.\n", strerror (mutex_error));
          free (status);
          status = NULL;
          break;
        }
        if (kill (analyseproc->pid, SIGSTOP)<0){
          if ((mutex_error = pthread_mutex_unlock(&(analyseproc->mutex_sigstop_sent))) != 0){
            fprintf (stderr, "analyser_thread_signalsender - pthread_mutex_unlock error: %s.\n", strerror (mutex_error));
          }
          perror ("analyser_thread_signalsender - kill (pid,SIGSTOP) error");
          free (status);
          status = NULL;
          break;
        }
        analyseproc->sigstop_sent = TRUE;
        if (analyseproc->debug)
          fprintf (stderr, "\n%u:analyser_thread_signalsender - SIGSTOP sent\n", analyseproc->pid);
        if ((mutex_error = pthread_mutex_unlock(&(analyseproc->mutex_sigstop_sent))) != 0){
          fprintf (stderr, "analyser_thread_signalsender - pthread_mutex_unlock error: %s.\n", strerror (mutex_error));
          free (status);
          status = NULL;
          break;
        }

        if ((mutex_error = pthread_mutex_lock(&(analyseproc->mutex_i))) != 0){
          fprintf (stderr, "analyser_thread_signalsender - pthread_mutex_lock error: %s.\n", strerror (mutex_error));
          free (status);
```

```
        status = NULL;
148     break;
      }
      while (analyseproc->change_i == 0){
        if (analyseproc->debug)
          fprintf (stderr, "\n%u:analyser_thread_signalsender - before pthread_cond_wait\n", analyseproc->pid);
        if ((cond_error = pthread_cond_wait(&(analyseproc->cond_i), &(analyseproc->mutex_i))) != 0){
          fprintf (stderr, "analyser_thread_signalsender - pthread_cond_wait error: %s.\n", strerror (cond_error));
          free (status);
          status = NULL;
          break;
        }
      }
      if (status == NULL){
        break;
      }
      analyseproc->change_i--;
      if (analyseproc->debug)
        fprintf (stderr, "\n%u:analyser_thread_signalsender - after pthread_cond_wait change_i:%u\n", analyseproc->pid,
      if ((mutex_error = pthread_mutex_unlock(&(analyseproc->mutex_i))) != 0){
        fprintf (stderr, "analyser_thread_signalsender - pthread_mutex_unlock error: %s.\n", strerror (mutex_error));
        free (status);
        status = NULL;
        break;
      }

      if (analyseproc->debug){
        if (analyseproc->i == 2){
          before_sigstop_tv = tempo_inicial_tv;
        } else {
          before_sigstop_tv = after_sigstop_tv;
        }
        if (gettimeofday (&after_sigstop_tv, NULL) < 0){
          perror ("analyser_thread_signalsender - gettimeofday error");
          free (status);
          status = NULL;
          break;
        }
185
        /* time between SIGSTOPS */
        if (timeval_subtract (&between_sigstops_tv, after_sigstop_tv, before_sigstop_tv, analyseproc->debug) != 0){
          perror ("analyser_thread_signalsender - timeval_subtract error");
          free (status);
          status = NULL;
          break;
        }
        fprintf (stderr, "\n%u:analyser_thread_signalsender - sigstop sent i:%lu\n", analyseproc->pid, analyseproc->i);
        fprintf (stderr, "\n%u:analyser_thread_signalsender - time between SIGSTOPS:%.15fs\n", analyseproc->pid, (betwee
      }
    } /* while (TRUE) */

    if (analyseproc->debug)
      fprintf (stderr, "\n%u:analyser_thread_signalsender - end of send SIGSTOP thread\n", analyseproc->pid);

    if (status == NULL){
      if ((mutex_error = pthread_mutex_lock(&(analyseproc->mutex_analyser_thread_signalsender_error))) != 0){
        fprintf (stderr, "analyser_thread_signalsender - pthread_mutex_lock error: %s.\n", strerror (mutex_error));
      }
      analyseproc->analyser_thread_signalsender_error = TRUE;
      if ((mutex_error = pthread_mutex_unlock(&(analyseproc->mutex_analyser_thread_signalsender_error))) != 0){
        fprintf (stderr, "analyser_thread_signalsender - pthread_mutex_unlock error: %s.\n", strerror (mutex_error));
      }
    }
    return status;
  }

  int
  process_detach(element_process_detach pd)
  {
    int error = 0, status_wait, status = 0;

    /*
      Linux wrongly insists the child be stopped
```

```
    before detaching.  Arghh.  We go through hoops
    to make a clean break of things.
222 */

  if ((error = ptrace (PTRACE_DETACH, pd.pid, (char *) 1, pd.sig)) == 0) {
    /* On a clear day, you can see forever. */
  }
  else if (errno != ESRCH) {
    /* Shouldn't happen. */
    fprintf (stderr, "process_detach - ptrace (PTRACE_DETACH, %u,... ) error: %s.\n", pd.pid, strerror (errno));
    status = -1;
  }
  else if (kill(pd.pid, 0) < 0) {
    if (errno != ESRCH)
      perror("process_detach - checking sanity error");
    status = -1;
  }
  else if (kill(pd.pid, SIGSTOP) < 0) {
    if (errno != ESRCH)
      perror("process_detach - stopping child error");
    status = -1;
  }
  else {
    for (;;) {
      if (waitpid(pd.pid, &status_wait, 0) < 0) {
        if (errno != ECHILD)
          perror("process_detach - waiting error");
        status = -1;
        break;
      }
      if (!WIFSTOPPED(status_wait)) {
        break;
      }
      if (WSTOPSIG(status_wait) == SIGSTOP) {
        if ((error = ptrace (PTRACE_DETACH, pd.pid, (char *) 1, pd.sig)) < 0) {
          if (errno != ESRCH)
            fprintf (stderr, "process_detach - ptrace (PTRACE_DETACH, %u,... ) error: %s.\n", pd.pid, strerror (errno)
          status = -1;
        }
259     break;
      }
      if ((error = ptrace (PTRACE_CONT, pd.pid, (char *) 1,  WSTOPSIG(status_wait) == SIGTRAP ? 0 : WSTOPSIG(status_wa
        if (errno != ESRCH)
          fprintf (stderr, "process_detach - ptrace (PTRACE_CONT, %u,... ) error: %s.\n", pd.pid, strerror (errno));
        status = -1;
        break;
      }
    }
  }

  return 0;
}

int
upeek(int pid, long int off, long int *res)
{
  long val;
  errno = 0;
  val = ptrace (PTRACE_PEEKUSER, pid, (char *) off, 0);
  if (val == -1 && errno) {
    fprintf (stderr, "upeek - ptrace (PTRACE_PEEKUSER, %u,... ) error: %s.\n", pid, strerror(errno));
    return -1;
  }
  *res = val;
  return 0;
}

void
*analyser (void * void_analyseproc)
{
  /* structures for process resources */
  struct rusage total_wait, final_getrusage;
  struct timeval tempo_inicial_tv, tempo_final_tv, momento_ins_inicial_tv, momento_ins_abs_tv;
```

```
        double momento_ins = 0;
        struct timeval tempo_decorrido_tv, real_time_tv, momento_ins_tv;
        /* calculating process start_time, real run time and effective run time */
296     struct timeval effective_run_time, effective_run_time_ant, effective_run_time_diff;
        double process_start_time = 0, real_run_time = 0;
        struct timeval total_effective_run_time;
        registro swap_vector[SWAPVECTORSIZE];

        /* control variables */
        pid_t ffeax = -1;
        pthread_t analisador_tid = 0;
        int *thread_return = NULL;
        int thread_error = 0;
        int mutex_error;
        int cond_error;
        pointer_thread_ffork analyseproc = NULL;
        pointer_thread_analyser sigsendproc = NULL;
        pthread_attr_t fftattr;
        pthread_t fftidvector[MAXFORKSTOFOLLOW];
        unsigned int fftid_number = 0;
        int *status = NULL;
        int status_wait4;
        FILE *proc_process_statf;
        char *proc_process_stat = NULL;
        char tmpstring[MAXPATHLEN];
        char string_pid[N_PROC_DIGIT];
        int fscanf_error;
        unsigned long int i;
        unsigned int j, k, l, m;
        int sig = -1;
        unsigned int interval_fforks = 0;
        /* system calls counter */
        unsigned long int sys_call_in_cont = 0;
        unsigned long int sys_call_no[NO_SYS_CALLS_PLUSONE];
        /* number and return of system call */
        long int scno=0, eax=-1;
        /* reading of /proc/PID/stat */
        int proc_starttime;
        unsigned int proc_vsize;
        unsigned int proc_rss;
333     int proc_void_int;
        unsigned int proc_void_uint;
        char proc_void_char;
        /* detaching process data */
        element_process_detach pd;
        /* waiting for follow forks threads end */
        unsigned int trylock_mutex_timeout;

        /* structures initialization */
        if (gettimeofday (&momento_ins_abs_tv, NULL) < 0){
          perror ("analyser - gettimeofday error");
          free (status);
          status = NULL;
          return status;
        }
        tempo_decorrido_tv.tv_sec = 0;
        tempo_decorrido_tv.tv_usec = 0;
        momento_ins_tv.tv_sec = 0;
        momento_ins_tv.tv_usec = 0;
        effective_run_time_ant.tv_sec = 0;
        effective_run_time_ant.tv_usec = 0;
        for (l = 0; l < MAXFORKSTOFOLLOW; l++){
          fftidvector[l] = 0;
        }
        l = 0;
        for (i = 0; i < NO_SYS_CALLS_PLUSONE; sys_call_no[i++] = 0);

        analyseproc = void_analyseproc;
        if ((status = (int *) malloc (sizeof(int)))!=NULL){
          *status = 1;
        }else{
          perror ("analyser - malloc error");
          if (analyseproc->flag & READFSWAPL){
```

```
    if ((mutex_error = pthread_mutex_lock(&(analyseproc->mutex_ffanalyser_thread_end))) != 0){
      fprintf (stderr, "analyser - pthread_mutex_lock error: %s.\n", strerror (mutex_error));
    }
    analyseproc->ffanalyser_thread_end = TRUE;
370 if ((mutex_error = pthread_mutex_unlock(&(analyseproc->mutex_ffanalyser_thread_end))) != 0){
      fprintf (stderr, "analyser - pthread_mutex_unlock error: %s.\n", strerror (mutex_error));
    }
  }

  return status;
}

/*
  Analyser algorithm
 */

/* SIGSTOP sender thread */
if ((sigsendproc = (pointer_thread_analyser) malloc (sizeof(element_thread_analyser)))!=NULL){
  pthread_cond_init(&(sigsendproc->cond_i), NULL);
  pthread_mutex_init(&(sigsendproc->mutex_i), NULL);
  sigsendproc->change_i = 0;
  sigsendproc->i = 0;
  pthread_mutex_init(&(sigsendproc->mutex_sigstop_sent), NULL);
  sigsendproc->sigstop_sent = FALSE;
  pthread_mutex_init(&(sigsendproc->mutex_analyser_thread_signalsender_error), NULL);
  sigsendproc->analyser_thread_signalsender_error = FALSE;
  sigsendproc->pid = analyseproc->pid;
  sigsendproc->flag = analyseproc->flag;
  sigsendproc->seconds = analyseproc->seconds;
  sigsendproc->useconds = analyseproc->useconds;
  sigsendproc->tam_loop = analyseproc->tam_loop;
  sigsendproc->debug = analyseproc->debug;
} else {
  perror ("analyser - malloc of analyserproc error");
  if (analyseproc->flag & READFSWAPL){
    if ((mutex_error = pthread_mutex_lock(&(analyseproc->mutex_ffanalyser_thread_end))) != 0){
      fprintf (stderr, "analyser - pthread_mutex_lock error: %s.\n", strerror (mutex_error));
    }
    analyseproc->ffanalyser_thread_end = TRUE;
    if ((mutex_error = pthread_mutex_unlock(&(analyseproc->mutex_ffanalyser_thread_end))) != 0){
      fprintf (stderr, "analyser - pthread_mutex_unlock error: %s.\n", strerror (mutex_error));
407 }
  }

  free (status);
  status = NULL;
  return status;
}
if ((thread_error = pthread_create(&analisador_tid, NULL, analyser_thread_signalsender, sigsendproc)) != 0) {
  if (thread_error == EAGAIN)
    fprintf (stderr, "analyser - pthread_create: not enough system resources to create a process for the new thread,
  if (thread_error == EINVAL)
    fprintf (stderr, "analyser - pthread_create: invalid thread attributes.\n");
  fprintf (stderr, "analyser - pthread_create error: %s.\n", strerror(thread_error));
  if (analyseproc->flag & READFSWAPL){
    if ((mutex_error = pthread_mutex_lock(&(analyseproc->mutex_ffanalyser_thread_end))) != 0){
      fprintf (stderr, "analyser - pthread_mutex_lock error: %s.\n", strerror (mutex_error));
    }
    analyseproc->ffanalyser_thread_end = TRUE;
    if ((mutex_error = pthread_mutex_unlock(&(analyseproc->mutex_ffanalyser_thread_end))) != 0){
      fprintf (stderr, "analyser - pthread_mutex_unlock error: %s.\n", strerror (mutex_error));
    }
  }

  free (status);
  status = NULL;
  return status;
}

if (analyseproc->debug)
  fprintf (stderr, "\n%u:analyser - analisador_tid:%lu\n", analyseproc->pid, analisador_tid);

/* attach panalyser to the process */
```

```
\
if ((analyseproc->flag & ATTACHED) || (analyseproc->flag & FOLLOWFORK)){
  if (ptrace (PTRACE_ATTACH, analyseproc->pid, (char *) 1, 0) < 0){
    if ((thread_error = pthread_cancel (analisador_tid)) != 0) {
      fprintf (stderr, "analyser - pthread_cancel error: %s - thread id:%lu.\n", strerror(thread_error), analisador_
    }
444 if ((thread_error = pthread_join(analisador_tid, (void **) thread_return)) != 0){
      fprintf (stderr, "analyser - pthread_join error: %s.\n", strerror(thread_error));
    }
    fprintf (stderr, "analyser - ptrace (PTRACE_ATTACH, %u,... ) error: %s.\n", analyseproc->pid, strerror(errno));
    if (analyseproc->flag & READFSWAPL){
      if ((mutex_error = pthread_mutex_lock(&(analyseproc->mutex_ffanalyser_thread_end))) != 0){
        fprintf (stderr, "analyser - pthread_mutex_lock error: %s.\n", strerror (mutex_error));
      }
      analyseproc->ffanalyser_thread_end = TRUE;
      if ((mutex_error = pthread_mutex_unlock(&(analyseproc->mutex_ffanalyser_thread_end))) != 0){
        fprintf (stderr, "analyser - pthread_mutex_unlock error: %s.\n", strerror (mutex_error));
      }
    }
    free (status);
    status = NULL;
    return status;
  }
}

/*
  From this point, if some error occur it is necessary to detach from the
  analysed process.
 */

/*
   Open /proc/PID/stat file
*/
memset(string_pid, '\0', sizeof (string_pid));
sprintf(string_pid, "%u", (unsigned int) analyseproc->pid);
memset(tmpstring, '\0', sizeof (tmpstring));
if ((strlen("/proc/") + strlen("/stat")) < (MAXPATHLEN - strlen(string_pid))){
  strcat(tmpstring, "/proc/");
  strcat(tmpstring, string_pid);
  proc_process_stat = strdup (strcat (tmpstring, "/stat"));
  if ((proc_process_statf = fopen(proc_process_stat, "r")) == NULL){
    fprintf(stderr, "analyser - can't fopen '%s': %s\n", proc_process_stat, strerror(errno));
  }
481 }

if (analyseproc->debug)
  fprintf (stderr, "\n%u: analyser\n", analyseproc->pid);

/*
  Initialize the flag variables to make the first SIGTRAP received
  an alert of get in a SYSCALL. Now on the process is being ptraced.
  Initialize the flag_swap to indicate the opening of swap file.
*/
analyseproc->flag |= (INSYSCALL | STARTUP);
analyseproc->flag_swap |= FIRSTSWAP;
i=0;
j=0;
while ((i <= analyseproc->tam_loop) || (analyseproc->flag & UNTILEND)){
  /*
     If it is the first time use gettimeofday for tempo_inicial.
  */
  if (analyseproc->flag & STARTUP){
    if (gettimeofday (&tempo_inicial_tv, NULL) < 0){
      perror ("analyser - gettimeofday error");
      free (status);
      status = NULL;
      break;
    }
    momento_ins_inicial_tv = tempo_inicial_tv;
    if (analyseproc->debug)
      fprintf (stderr, "\n%u:analyser - STARTUP start time:%fus\n", analyseproc->pid, (((double) tempo_inicial_tv.tv
  }

  if ((mutex_error = pthread_mutex_lock(&(sigsendproc->mutex_analyser_thread_signalsender_error))) != 0){
```

```
      fprintf (stderr, "analyser - pthread_mutex_lock error: %s.\n", strerror (mutex_error));
      free (status);
      status = NULL;
      break;
    }
    if (sigsendproc->analyser_thread_signalsender_error == TRUE){
518   if ((mutex_error = pthread_mutex_unlock(&(sigsendproc->mutex_analyser_thread_signalsender_error))) != 0){
        fprintf (stderr, "analyser - pthread_mutex_unlock error: %s.\n", strerror (mutex_error));
      }
      free (status);
      status = NULL;
      break;
    }
    if ((mutex_error = pthread_mutex_unlock(&(sigsendproc->mutex_analyser_thread_signalsender_error))) != 0){
      fprintf (stderr, "analyser - pthread_mutex_unlock error: %s.\n", strerror (mutex_error));
      free (status);
      status = NULL;
      break;
    }

    if (analyseproc->debug)
      fprintf (stderr, "\n%u:analyser - before wait4\n", analyseproc->pid);

    if (wait4 (analyseproc->pid, &status_wait4, WUNTRACED, &total_wait) != analyseproc->pid){
      perror ("analyser - wait4 error");
      free (status);
      status = NULL;
      break;
    }

    if (analyseproc->debug)
      fprintf (stderr, "\n%u:analyser - after wait4\n", analyseproc->pid);

    if (WIFSTOPPED(status_wait4)){
      sig = WSTOPSIG(status_wait4);

      if (analyseproc->debug)
        fprintf (stderr, "\n%u:analyser - the process which causes the return is stopped with the signal:%u\n", analys

      if (sig == SIGTRAP){

        if (analyseproc->debug)
          fprintf (stderr, "\n%u:analyser - SIGTRAP received\n", analyseproc->pid);
555
        if (analyseproc->flag & INSYSCALL){
          analyseproc->flag &= ~INSYSCALL;
          if (upeek(analyseproc->pid, 4*ORIG_EAX, &scno) < 0){
            perror ("analyser - upeek error");
            free (status);
            status = NULL;
            break;
          }

          if (analyseproc->debug)
            fprintf (stderr, "\n%u:analyser - INSYSCALL scno:%ld\n", analyseproc->pid, scno);

          if ((scno == __NR_fork) || (scno == __NR_vfork)){
            if (upeek(analyseproc->pid, 4*EAX, &eax) < 0){
              perror ("analyser - upeek error");
              free (status);
              status = NULL;
              break;
            }

            if (analyseproc->debug)
              fprintf (stderr, "\n%u:analyser - fork/vfork return:%ld\n", analyseproc->pid, eax);

          }/* if ((scno == __NR_fork) || (scno == __NR_vfork)) */

          if ((scno == __NR_execve) && (analyseproc->flag & WAITEXECVE)){
            analyseproc->flag &= ~WAITEXECVE;
          } else if ((scno == __NR_wait4) && !(analyseproc->flag & SUSPENDED)){
            analyseproc->flag |= SUSPENDED;
```

```
            sys_call_in_cont++;
            sys_call_no[scno]++;
          }else {
            analyseproc->flag &= ~SUSPENDED;
            sys_call_in_cont++;
            sys_call_no[scno]++;
          }
592     } else if (!(analyseproc->flag & INSYSCALL)){
          analyseproc->flag |= INSYSCALL;

          if ((scno == __NR_fork) || (scno == __NR_vfork)){
            if (eax < 0){
              if (upeek(analyseproc->pid, 4*EAX, &eax) < 0){
                perror ("analyser - upeek error");
                free (status);
                status = NULL;
                break;
              }

              if (analyseproc->debug)
                fprintf (stderr, "\n%u:analyser - fork/vfork return:%ld\n", analyseproc->pid, eax);

            }

            if ((analyseproc->flag & OPTFFORK) && (eax > 0) && (l < MAXFORKSTOFOLLOW)){

              if (interval_fforks == 0){
                interval_fforks = analyseproc->interval_fforks;

                if (analyseproc->debug)
                  fprintf (stderr, "\n%u:analyser - fork/vfork return:%ld\n", analyseproc->pid, eax);

                ffeax = (pid_t) eax;
                analyseproc->ffeaxvector[l] = ffeax;

                /*
                   Thread for children process of the analysed process.
                */
                if ((analyseproc->ffanalyseproc[l] = (pointer_thread_ffork) malloc (sizeof(element_thread_ffork)))!=NU
                  analyseproc->ffanalyseproc[l]->pid = ffeax;
                  analyseproc->ffanalyseproc[l]->flag = analyseproc->flag;
                  analyseproc->ffanalyseproc[l]->flag |= (READFSWAPL | FOLLOWFORK);
                  analyseproc->ffanalyseproc[l]->flag &= ~OPTFFORK;
                  analyseproc->ffanalyseproc[l]->flag_swap = analyseproc->flag_swap;
629               analyseproc->ffanalyseproc[l]->seconds = analyseproc->seconds;
                  analyseproc->ffanalyseproc[l]->useconds = analyseproc->useconds;
                  analyseproc->ffanalyseproc[l]->tam_loop = (analyseproc->tam_loop - i + 1);
                  analyseproc->ffanalyseproc[l]->outf = NULL;
                  analyseproc->ffanalyseproc[l]->outfname = NULL;
                  analyseproc->ffanalyseproc[l]->stf_rusage = NULL;
                  analyseproc->ffanalyseproc[l]->stfname_rusage = NULL;
                  analyseproc->ffanalyseproc[l]->stf_syscall = NULL;
                  analyseproc->ffanalyseproc[l]->stfname_syscall = NULL;
                  if (analyseproc->swap_outfname){
                    analyseproc->ffanalyseproc[l]->swap_outfname = strdup(analyseproc->swap_outfname);
                  } else {
                    analyseproc->ffanalyseproc[l]->swap_outfname = NULL;
                  }
                  analyseproc->ffanalyseproc[l]->criterio_ord = analyseproc->criterio_ord;
                  analyseproc->ffanalyseproc[l]->process_start_time = 0;
                  analyseproc->ffanalyseproc[l]->panalyser_start_time = 0;
                  analyseproc->ffanalyseproc[l]->real_run_time = 0;
                  analyseproc->ffanalyseproc[l]->effective_run_time.tv_sec = 0;
                  analyseproc->ffanalyseproc[l]->effective_run_time.tv_usec = 0;
                  pthread_mutex_init(&(analyseproc->ffanalyseproc[l]->mutex_ffanalyser_thread_end), NULL);
                  analyseproc->ffanalyseproc[l]->ffanalyser_thread_end = FALSE;
                  analyseproc->ffanalyseproc[l]->debug = analyseproc->debug;

                  if (pthread_attr_init(&fftattr) != 0){
                    fprintf (stderr, "\nanalyser - pthread_attr_init.\n");
                  }

                  if ((thread_error = pthread_attr_setdetachstate(&fftattr, PTHREAD_CREATE_DETACHED)) !=0 ){
```

```
                fprintf (stderr, "analyser - pthread_attr_setdetachstate error: %s.\n", strerror (thread_error));
              }

              if ((thread_error = pthread_create(&(fftidvector[l++]), &fftattr, analyser, analyseproc->ffanalysepr
                if (thread_error == EAGAIN)
                  fprintf (stderr, "analyser - pthread_create: not enough system resources to create a process for
                if (thread_error == EINVAL)
                  fprintf (stderr, "analyser - pthread_create: invalid thread attributes.\n");
666             fprintf (stderr, "analyser - pthread_create error: %s.\n", strerror(thread_error));
                analyseproc->ffeaxvector[l] = 0;
              }
              fftid_number = l;
              analyseproc->ffeax_number = l;

              if (analyseproc->debug)
                fprintf (stderr, "\n%u:analyser - thread id:%lu\n", analyseproc->ffanalyseproc[l-1]->pid, fftidvec

            } /* if ((analyseproc->ffanalyseproc[l] = (pointer_thread_ffork) malloc (sizeof(element_thread_ffork))
            else{
              analyseproc->ffeaxvector[l] = 0;
            }
          } /* if (interval_fforks == 0) */
          interval_fforks--;

        } /* if ((analyseproc->flag & OPTFFORK) && (eax > 0) && (l < MAXFORKSTOFOLLOW)) */
        else {
          if (l >= MAXFORKSTOFOLLOW)
            fprintf (stderr, "analyser - Maximum number of forks to follow reached, no more followin forks.\n");
          if (eax <= 0)
            fprintf (stderr, "analyser - Invalid PID value read from fork syscall.\n");
        }
      }/* if ((scno == __NR_fork) || (scno == __NR_vfork)) */

      if (analyseproc->debug)
        fprintf (stderr, "\n%u:analyser - OUTSYSCALL scno:%ld\n", analyseproc->pid, scno);

    } /* if (!(flag & INSYSCALL)) */
  } /* if (sig == SIGTRAP) */

  /*
    If the signal is a SIGSTOP and analyser->sigstop_sent == TRUE
    or the first loop, save results.
  */

  if ((mutex_error = pthread_mutex_lock(&(sigsendproc->mutex_sigstop_sent))) != 0){
703 fprintf (stderr, "analyser - pthread_mutex_lock error: %s.\n", strerror (mutex_error));
    free (status);
    status = NULL;
    break;
  }

  if ((mutex_error = pthread_mutex_lock(&(sigsendproc->mutex_i))) != 0){
    fprintf (stderr, "analyser - pthread_mutex_lock error: %s.\n", strerror (mutex_error));
    free (status);
    status = NULL;
    break;
  }

  if (((sig == SIGSTOP) && (sigsendproc->sigstop_sent == TRUE)) || (analyseproc->flag & STARTUP)){

    /*
      Calculate interval betwewn SIGSTOPs.
    */
    if (gettimeofday (&tempo_final_tv, NULL) < 0){
      perror ("analyser - gettimeofday error");
      free (status);
      status = NULL;
      break;
    }
    momento_ins_abs_tv = tempo_final_tv;

    if (analyseproc->debug)
      fprintf (stderr, "\n%u:analyser - momento_ins_abs_tv:%1.15fs\n", analyseproc->pid, (momento_ins_abs_tv.tv_se
```

```
/* time between SIGSTOPS */
if (timeval_subtract (&tempo_decorrido_tv, tempo_final_tv, tempo_inicial_tv, analyseproc->debug) != 0){
  perror ("analyser - timeval_subtract error");
  free (status);
  status = NULL;
  break;
}

740  /* saved data on the swap_vector */
if (timeval_subtract (&momento_ins_tv, momento_ins_abs_tv, momento_ins_inicial_tv, analyseproc->debug) != 0){
  perror ("analyser - timeval_subtract error");
  free (status);
  status = NULL;
  break;
}
momento_ins = momento_ins_tv.tv_sec + ((double) momento_ins_tv.tv_usec) / M;

if (analyseproc->debug){
  fprintf (stderr, "\n%u:analyser - end time:%fus\n", analyseproc->pid, (((double) tempo_final_tv.tv_sec * M)
  fprintf (stderr, "%u:analyser - time between SIGSTOPS:%fus\n", analyseproc->pid, (((double) tempo_decorrido_
  fprintf (stderr, "%u:analyser - measurement time:%fus\n", analyseproc->pid, (((double) momento_ins_tv.tv_sec
}
/* End of SIGSTOPS interval calc */

if (analyseproc->debug)
  fprintf (stderr, "\n%u:analyser - i:%lu\n", analyseproc->pid, i);
if ((sig == SIGSTOP) && (sigsendproc->sigstop_sent == TRUE)){
  if (sigsendproc->change_i == 0){
    if (analyseproc->debug)
      fprintf (stderr, "\n%u:analyser - before pthread_cond_signal\n", analyseproc->pid);
    if ((cond_error = pthread_cond_signal(&(sigsendproc->cond_i))) != 0){
      fprintf (stderr, "analyser - pthread_cond_signal error: %s.\n", strerror (cond_error));
      free (status);
      status = NULL;
      break;
    }
  }
  sigsendproc->change_i++;
  if (analyseproc->debug)
    fprintf (stderr, "\n%u:analyser - after pthread_cond_signal change_i:%u\n", analyseproc->pid, sigsendproc-
}

if (sigsendproc->change_i > 1){
  if ((mutex_error = pthread_mutex_unlock(&(sigsendproc->mutex_i))) != 0){
  fprintf (stderr, "analyser - pthread_mutex_unlock error: %s.\n", strerror (mutex_error));
777  }
  if ((mutex_error = pthread_mutex_unlock(&(sigsendproc->mutex_sigstop_sent))) != 0){
  fprintf (stderr, "analyser - pthread_mutex_unlock error: %s.\n", strerror (mutex_error));
  }
  fprintf (stderr, "analyser: more waits then sigstops error.\n");
  free (status);
  status = NULL;
  break;
}

real_time_tv = tempo_decorrido_tv;
tempo_decorrido_tv.tv_sec = 0;
tempo_decorrido_tv.tv_usec = 0;
/* If it isn't the first time tempo_inicial = tempo_final. */
if (!(analyseproc->flag & STARTUP)){
  tempo_inicial_tv = tempo_final_tv;
  if (analyseproc->debug)
    fprintf (stderr, "\n%u:analyser - start time:%fus\n", analyseproc->pid, (((double) tempo_inicial_tv.tv_se
}
analyseproc->flag |= SAVERESULT;
sigsendproc->sigstop_sent = FALSE;
sigsendproc->i = ++i;

} /* if (((sig == SIGSTOP) && (sigsendproc->sigstop_sent == TRUE)) || (analyseproc->flag & STARTUP)) */
if ((mutex_error = pthread_mutex_unlock(&(sigsendproc->mutex_i))) != 0){
  fprintf (stderr, "analyser - pthread_mutex_unlock error: %s.\n", strerror (mutex_error));
  free (status);
```

```
        status = NULL;
        break;
      }
      if ((mutex_error = pthread_mutex_unlock(&(sigsendproc->mutex_sigstop_sent))) != 0){
        fprintf (stderr, "analyser - pthread_mutex_unlock error: %s.\n", strerror (mutex_error));
        free (status);
        status = NULL;
        break;
      }
      if ((analyseproc->flag & STARTUP) || ((i > analyseproc->tam_loop) && !(analyseproc->flag & UNTILEND))){
814
        /*
           Reading /proc/PID/stat.
        */
        if (proc_process_statf != NULL){
          if (fseek (proc_process_statf, 0, SEEK_SET) < 0){
            fprintf (stderr, "analyser - fseek error on file: %s.\n", proc_process_stat);
          } else {
            if (((fscanf_error = fscanf (proc_process_statf, "%d %s %c %d %d %d %d %u %u %u %u %u %d %d %d %d %d %d
              fprintf (stderr, "analyser - fscanf error on file: %s.\n", proc_process_stat);
            } else {
              if (analyseproc->debug)
                fprintf (stderr, "\n%u:analyser - proc_starttime:%d - proc_vsize:%u - proc_rss:%u\n", analyseproc->pid
            }
          }
        }
        /*
          End of reading /proc/PID/stat.
        */

        if (gen_proc_summary (total_wait.ru_utime, total_wait.ru_stime, momento_ins_abs_tv, proc_starttime, &process_s
          fprintf (stderr, "%u:analyser - gen_proc_summary error: %s.\n", analyseproc->pid, strerror (errno));
        }
        if ((i > analyseproc->tam_loop) && !(analyseproc->flag & UNTILEND)){
          analyseproc->process_start_time = (time_t) process_start_time;
          analyseproc->panalyser_start_time = (time_t) (momento_ins_inicial_tv.tv_sec + ((double)momento_ins_inicial_t
          analyseproc->real_run_time = real_run_time;
          analyseproc->effective_run_time = total_effective_run_time;
        }
      } /* if ((analyseproc->flag & STARTUP) || ((i == analyseproc->tam_loop) && !(analyseproc->flag & UNTILEND))) */
    } /* if (WIFSTOPPED(status_wait4)) */
    else if (WIFEXITED(status_wait4)){

      if (getrusage (RUSAGE_CHILDREN, &final_getrusage) < 0){
        perror ("analyser - getrusage error");
        free (status);
        status = NULL;
851     break;
      }

      if (gettimeofday (&momento_ins_abs_tv, NULL) < 0){
        perror ("analyser - gettimeofday error");
        free (status);
        status = NULL;
        return status;
      }

      if (timeval_subtract (&momento_ins_tv, momento_ins_abs_tv, momento_ins_inicial_tv, analyseproc->debug) != 0){
        perror ("analyser - timeval_subtract error");
        free (status);
        status = NULL;
        break;
      }
      momento_ins = momento_ins_tv.tv_sec + ((double) momento_ins_tv.tv_usec) / M;

      /*
         Reading /proc/PID/stat.
      */
      if (proc_process_statf != NULL){
        if (fseek (proc_process_statf, 0, SEEK_SET) < 0){
          fprintf (stderr, "analyser - fseek error on file: %s.\n", proc_process_stat);
        } else {
          if (((fscanf_error = fscanf (proc_process_statf, "%d %s %c %d %d %d %d %d %u %u %u %u %u %d %d %d %d %d %d
```

```
            fprintf (stderr, "analyser - fscanf error on file: %s.\n", proc_process_stat);
          } else {
            if (analyseproc->debug)
              fprintf (stderr, "\n%u:analyser - proc_starttime:%d - proc_vsize:%u - proc_rss:%u\n", analyseproc->pid,
          }
        }
      }
      /*
        End of reading /proc/PID/stat.
      */

888   analyseproc->flag_swap &= ~RESULTSWAP;
      swap_vector[j].indice = i;
      swap_vector[j].momento_insercao = momento_ins;
      swap_vector[j].sys_res_usage = final_getrusage;
      swap_vector[j].proc_stat.starttime = proc_starttime;
      swap_vector[j].proc_stat.vsize = proc_vsize;
      swap_vector[j].proc_stat.rss = proc_rss;
      swap_vector[j].sys_call_in_cont = sys_call_in_cont;

      if (gen_proc_summary (final_getrusage.ru_utime, final_getrusage.ru_stime, momento_ins_abs_tv, proc_starttime, &p
        fprintf (stderr, "%u:analyser - gen_proc_summary error: %s.\n", analyseproc->pid, strerror (errno));
      }
      analyseproc->process_start_time = (time_t) process_start_time;
      analyseproc->panalyser_start_time = (time_t) (momento_ins_inicial_tv.tv_sec + ((double)momento_ins_inicial_tv.tv
      analyseproc->real_run_time = real_run_time;
      analyseproc->effective_run_time = total_effective_run_time;

      if (timeval_add (&effective_run_time, final_getrusage.ru_stime, final_getrusage.ru_utime, analyseproc->debug) !=
        perror ("analyser - timeval_add error");
      }
      if (analyseproc->debug)
        fprintf (stderr, "\n%u:analyser exit - real_run_time:%1.10fs - effective_run_time:%1.10f\n", analyseproc->pid,
      swap_vector[j].proc_stat.cpu_percentage = (effective_run_time.tv_sec + ((double) effective_run_time.tv_usec) / M

      for (k=0; k < NO_SYS_CALLS_PLUSONE; k++){
        swap_vector[j].sys_call_no[k] = sys_call_no[k];
      }
      j++;

      analyseproc->flag |= EXITING;
      fprintf (stderr, "\n%u:analyser - the process which causes the return ended normaly with status:%d\n", analysepr
      break;
    }
    else if (WIFSIGNALED(status_wait4)){

      if (getrusage (RUSAGE_CHILDREN, &final_getrusage) < 0){
        perror ("analyser - getrusage error");
925     free (status);
        status = NULL;
        break;
      }

      if (gettimeofday (&momento_ins_abs_tv, NULL) < 0){
        perror ("analyser - gettimeofday error");
        free (status);
        status = NULL;
        return status;
      }

      if (timeval_subtract (&momento_ins_tv, momento_ins_abs_tv, momento_ins_inicial_tv, analyseproc->debug) != 0){
        perror ("analyser - timeval_subtract error");
        free (status);
        status = NULL;
        break;
      }
      momento_ins = momento_ins_tv.tv_sec + ((double) momento_ins_tv.tv_usec) / M;

      /*
         Reading /proc/PID/stat.
      */
      if (proc_process_statf != NULL){
        if (fseek (proc_process_statf, 0, SEEK_SET) < 0){
```

```
        fprintf (stderr, "analyser - fseek error on file: %s.\n", proc_process_stat);
      } else {
        if (((fscanf_error = fscanf (proc_process_statf, "%d %s %c %d %d %d %d %d %u %u %u %u %u %d %d %d %d %d %d %
          fprintf (stderr, "analyser - fscanf error on file: %s.\n", proc_process_stat);
        } else {
          if (analyseproc->debug)
            fprintf (stderr, "\n%u:analyser - proc_starttime:%d - proc_vsize:%u - proc_rss:%u\n", analyseproc->pid,
        }
      }
    }
    /*
      End of reading /proc/PID/stat.
962 */

    analyseproc->flag_swap &= ~RESULTSWAP;
    swap_vector[j].indice = i;
    swap_vector[j].momento_insercao = momento_ins;
    swap_vector[j].sys_res_usage = final_getrusage;
    swap_vector[j].proc_stat.starttime = proc_starttime;
    swap_vector[j].proc_stat.vsize = proc_vsize;
    swap_vector[j].proc_stat.rss = proc_rss;
    swap_vector[j].sys_call_in_cont = sys_call_in_cont;

    if (gen_proc_summary (final_getrusage.ru_utime, final_getrusage.ru_stime, momento_ins_abs_tv, proc_starttime, &p
      fprintf (stderr, "%u:analyser - gen_proc_summary error: %s.\n", analyseproc->pid, strerror (errno));
    }
    analyseproc->process_start_time = (time_t) process_start_time;
    analyseproc->panalyser_start_time = (time_t) (momento_ins_inicial_tv.tv_sec + ((double)momento_ins_inicial_tv.tv
    analyseproc->real_run_time = real_run_time;
    analyseproc->effective_run_time = total_effective_run_time;

    if (timeval_add (&effective_run_time, final_getrusage.ru_stime, final_getrusage.ru_utime, analyseproc->debug) !=
      perror ("analyser - timeval_add error");
    }
    if (analyseproc->debug)
      fprintf (stderr, "\n%u:analyser exit - real_run_time:%1.10fs - effective_run_time:%1.10f\n", analyseproc->pid,
    swap_vector[j].proc_stat.cpu_percentage = (effective_run_time.tv_sec + ((double) effective_run_time.tv_usec) / 1

    for (k=0; k < NO_SYS_CALLS_PLUSONE; k++){
      swap_vector[j].sys_call_no[k] = sys_call_no[k];
    }
    j++;

    analyseproc->flag |= EXITING;
    fprintf (stderr, "\n%u:analyser - the process which causes the return ended with a non-detected signal:%d\n", a
    break;
  }

  if (ptrace (PTRACE_SYSCALL, analyseproc->pid, (char *) 1, sig == SIGTRAP ? 0 : sig) < 0){
999 fprintf (stderr, "analyser - ptrace (PTRACE_SYSCALL, %u,... ) error: %s.\n", analyseproc->pid, strerror (errno)
    free (status);
    status = NULL;
    break;
  }

  if (analyseproc->flag & SAVERESULT){
    analyseproc->flag &= ~SAVERESULT;
    analyseproc->flag_swap &= ~RESULTSWAP;
    swap_vector[j].indice = (i - 1);
    swap_vector[j].momento_insercao = momento_ins;
    swap_vector[j].sys_res_usage = total_wait;
    swap_vector[j].proc_stat.starttime = proc_starttime;
    swap_vector[j].proc_stat.vsize = proc_vsize;
    swap_vector[j].proc_stat.rss = proc_rss;
    swap_vector[j].sys_call_in_cont = sys_call_in_cont;

    if (timeval_add (&effective_run_time, total_wait.ru_stime, total_wait.ru_utime, analyseproc->debug) != 0){
      perror ("analyser - timeval_add error");
    }
    if (timeval_subtract (&effective_run_time_diff, effective_run_time, effective_run_time_ant, analyseproc->debug)
      perror ("analyser - timeval_subtract error");
    }
    if (analyseproc->debug)
```

```
        fprintf (stderr, "\n%u:analyser - real_time_tv:%1.10fs - effective_run_time_diff:%1.10f\n", analyseproc->pid,
      if (analyseproc->flag & STARTUP){
        analyseproc->flag &= ~STARTUP;
        swap_vector[j].proc_stat.cpu_percentage = 0;
      } else {
        swap_vector[j].proc_stat.cpu_percentage = (effective_run_time_diff.tv_sec + ((double) effective_run_time_diff.
      }
      effective_run_time_ant = effective_run_time;

      for (k=0; k < NO_SYS_CALLS_PLUSONE; k++){
        swap_vector[j].sys_call_no[k] = sys_call_no[k];
      }
      j++;
1036
      if (j == SWAPVECTORSIZE){
        if (data_swap_in (swap_vector, analyseproc->pid, analyseproc->swap_outfname, j, analyseproc->flag_swap, analys
          perror ("analyser - data_swap_in error");
          free (status);
          status = NULL;
          break;
        }
        j = 0;
        analyseproc->flag_swap |= RESULTSWAP;
        analyseproc->flag_swap &= ~FIRSTSWAP;
      }

    }/* if (analyseproc->flag & SAVERESULT) */

  } /* while ((i < tam_loop) || (analyseproc->flag & UNTILEND)) */

  if (analyseproc->debug){
    if (analyseproc->flag & EXITING){
      fprintf (stderr, "\n%u:analyser - process analysed ended\n", analyseproc->pid);
    }
    else {
      fprintf (stderr, "\n%u:analyser - process analysed NOT ended\n", analyseproc->pid);
    }
  }

  if (!(analyseproc->flag & EXITING) || (status == NULL)){
    pd.pid = analyseproc->pid;
    pd.sig = sig;
    if (process_detach (pd) < 0){
      fprintf (stderr, "\n%u:analyser - process_detach error.\n", analyseproc->pid);
    }
    /*
       If panalyser don't manage to detach from analysed process try to
       not interfere with the analysed process.
    */
    if (kill (analyseproc->pid, SIGCONT)<0){
1073  perror ("analyser - kill (analyseproc->pid,SIGCONT) error");
      free (status);
      status = NULL;
    }
  } /* if (!(analyseproc->flag & EXITING) || (status == NULL)) */

  if (analyseproc->flag & EXITING){

    if (analyseproc->debug)
      fprintf (stderr, "\n%u:analyser - analysed process has ended\n", analyseproc->pid);

    if (status != NULL){
      if ((thread_error = pthread_cancel (analisador_tid)) != 0) {
        fprintf (stderr, "analyser - pthread_cancel error: %s - thread id:%lu.\n", strerror(thread_error), analisador_
      }
    }
  }

  if (status == NULL){
    if ((thread_error = pthread_cancel (analisador_tid)) != 0) {
      fprintf (stderr, "analyser - pthread_cancel error: %s - thread id:%lu.\n", strerror(thread_error), analisador_ti
    }
    if ((thread_error = pthread_join(analisador_tid, (void **) thread_return)) != 0){
```

```
  fprintf (stderr, "analyser - pthread_join error: %s.\n", strerror(thread_error));
}

if (analyseproc->flag & OPTFFORK){
  for (l = 0; l < fftid_number; l++){
    if (fftidvector[l] > 0){
      trylock_mutex_timeout = 0;
      m = 0;
      while (TRUE){
        if ((mutex_error = pthread_mutex_trylock(&(analyseproc->ffanalyseproc[l]->mutex_ffanalyser_thread_end))) !
          if (mutex_error != EBUSY){
            fprintf (stderr, "analyser - pthread_mutex_lock error: %s.\n", strerror (mutex_error));
            break;
          }
        }
        if (mutex_error == EBUSY){
          if (trylock_mutex_timeout++ >= MUTEX_TRYL_TIMEOUT){
            break;
          } else {
            sleep (analyseproc->seconds);
            usleep (analyseproc->useconds);
          }
        }
        if (mutex_error == 0){
          if (analyseproc->ffanalyseproc[l]->ffanalyser_thread_end == TRUE){
            if ((mutex_error = pthread_mutex_unlock(&(analyseproc->ffanalyseproc[l]->mutex_ffanalyser_thread_end))
              fprintf (stderr, "analyser - pthread_mutex_unlock error: %s.\n", strerror (mutex_error));
            }
            break;
          }
          if ((mutex_error = pthread_mutex_unlock(&(analyseproc->ffanalyseproc[l]->mutex_ffanalyser_thread_end)))
            fprintf (stderr, "analyser - pthread_mutex_unlock error: %s.\n", strerror (mutex_error));
            break;
          }
          if (!(analyseproc->flag & UNTILEND)){
            if (m < (analyseproc->tam_loop - i + 1)){
              m++;
            }
            else{
              break;
            }
          }
          sleep (analyseproc->seconds);
          usleep (analyseproc->useconds);
        }/* if (mutex_error == 0) */
      }/* while (TRUE) */
      if ((mutex_error = pthread_mutex_destroy(&(analyseproc->ffanalyseproc[l]->mutex_ffanalyser_thread_end)))!= 0
        fprintf (stderr, "%u:analyser - pthread_mutex_destroy mutex_i error: %s.\n", analyseproc->pid, strerror(mu
      }
    }/* if (fftidvector[l] > 0) */
  }/* for (l = 0; l < fftid_number; l++) */
}/* if (analyseproc->flag & OPTFFORK) */

if (analyseproc->flag & OPTFFORK){
  if (pthread_attr_destroy(&fftattr) != 0){
    fprintf (stderr, "%u:analyser - pthread_attr_destroy error.\n", analyseproc->pid);
  }
}

if ((cond_error = pthread_cond_destroy(&(sigsendproc->cond_i)))!= 0){
  fprintf (stderr, "%u:analyser - pthread_cond_destroy error: %s.\n", analyseproc->pid, strerror(cond_error));
}
if ((mutex_error = pthread_mutex_destroy(&(sigsendproc->mutex_i)))!= 0){
  fprintf (stderr, "%u:analyser - pthread_mutex_destroy mutex_i error: %s.\n", analyseproc->pid, strerror(mutex_e
}
if ((mutex_error = pthread_mutex_destroy(&(sigsendproc->mutex_sigstop_sent)))!= 0){
  fprintf (stderr, "%u:analyser - pthread_mutex_destroy mutex_sigstop_sent error: %s.\n", analyseproc->pid, strer
}
if ((mutex_error = pthread_mutex_destroy(&(sigsendproc->mutex_analyser_thread_signalsender_error)))!= 0){
  fprintf (stderr, "%u:analyser - pthread_mutex_destroy mutex_analyser_thread_signalsender_error error: %s.\n", a
}

if (analyseproc->flag & READFSWAPL){
```

```
    if ((mutex_error = pthread_mutex_lock(&(analyseproc->mutex_ffanalyser_thread_end))) != 0){
      fprintf (stderr, "analyser - pthread_mutex_lock error: %s.\n", strerror (mutex_error));
    }
    analyseproc->ffanalyser_thread_end = TRUE;
    if ((mutex_error = pthread_mutex_unlock(&(analyseproc->mutex_ffanalyser_thread_end))) != 0){
      fprintf (stderr, "analyser - pthread_mutex_unlock error: %s.\n", strerror (mutex_error));
    }
  }

  return status;
}

if (!(analyseproc->flag_swap & RESULTSWAP)){
  if (data_swap_in (swap_vector, analyseproc->pid, analyseproc->swap_outfname, j, analyseproc->flag_swap, analysepro
    perror ("analyser - data_swap_in error");
1184  free (status);
    status = NULL;
  }
  analyseproc->flag_swap &= ~FIRSTSWAP;
}

if ((thread_error = pthread_join(analisador_tid, (void **) thread_return)) != 0){
  fprintf (stderr, "analyser - pthread_join error: %s.\n", strerror(thread_error));
  if (status != NULL){
    free (status);
    status = NULL;
  }
}

if (analyseproc->debug)
  fprintf (stderr, "\n%u:analyser - pthread_join of analyser_thread_signalsender\n", analyseproc->pid);


if (analyseproc->flag & OPTFFORK){
  for (l = 0; l < fftid_number; l++){
    if (fftidvector[l] > 0){
      trylock_mutex_timeout = 0;
      m = 0;
      while (TRUE){
        if ((mutex_error = pthread_mutex_trylock(&(analyseproc->ffanalyseproc[l]->mutex_ffanalyser_thread_end))) !=
          if (mutex_error != EBUSY){
            fprintf (stderr, "analyser - pthread_mutex_lock error: %s.\n", strerror (mutex_error));
            break;
          }
        }
        if (mutex_error == EBUSY){
          if (trylock_mutex_timeout++ >= MUTEX_TRYL_TIMEOUT){
            break;
          } else {
            sleep (analyseproc->seconds);
            usleep (analyseproc->useconds);
          }
1221    }
        if (mutex_error == 0){
          if (analyseproc->ffanalyseproc[l]->ffanalyser_thread_end == TRUE){
            if ((mutex_error = pthread_mutex_unlock(&(analyseproc->ffanalyseproc[l]->mutex_ffanalyser_thread_end)))
              fprintf (stderr, "analyser - pthread_mutex_unlock error: %s.\n", strerror (mutex_error));
            }
            break;
          }
          if ((mutex_error = pthread_mutex_unlock(&(analyseproc->ffanalyseproc[l]->mutex_ffanalyser_thread_end))) !=
            fprintf (stderr, "analyser - pthread_mutex_unlock error: %s.\n", strerror (mutex_error));
            break;
          }
          if (!(analyseproc->flag & UNTILEND)){
            if (m < (analyseproc->tam_loop - i + 1)){
              m++;
            }
            else{
              break;
            }
          }
          sleep (analyseproc->seconds);
```

```
            usleep (analyseproc->useconds);
          }/* if (mutex_error == 0) */
        }/* while (TRUE) */
        if ((mutex_error = pthread_mutex_destroy(&(analyseproc->ffanalyseproc[l]->mutex_ffanalyser_thread_end)))!= 0){
          fprintf (stderr, "%u:analyser - pthread_mutex_destroy mutex_i error: %s.\n", analyseproc->pid, strerror(mute
        }
      }/* if (fftidvector[l] > 0) */
    }/* for (l = 0; l < fftid_number; l++) */
  }/* if (analyseproc->flag & OPTFFORK) */


  if (analyseproc->flag & OPTFFORK){
    if (pthread_attr_destroy(&fftattr) != 0){
      fprintf (stderr, "%u:analyser - pthread_attr_destroy error.\n", analyseproc->pid);
    }
  }
1258
  if ((cond_error = pthread_cond_destroy(&(sigsendproc->cond_i)))!= 0){
    fprintf (stderr, "%u:analyser - pthread_cond_destroy error: %s.\n", analyseproc->pid, strerror(cond_error));
  }
  if ((mutex_error = pthread_mutex_destroy(&(sigsendproc->mutex_i)))!= 0){
    fprintf (stderr, "%u:analyser - pthread_mutex_destroy mutex_i error: %s.\n", analyseproc->pid, strerror(mutex_erro
  }
  if ((mutex_error = pthread_mutex_destroy(&(sigsendproc->mutex_sigstop_sent)))!= 0){
    fprintf (stderr, "%u:analyser - pthread_mutex_destroy mutex_sigstop_sent error: %s.\n", analyseproc->pid, strerror
  }
  if ((mutex_error = pthread_mutex_destroy(&(sigsendproc->mutex_analyser_thread_signalsender_error)))!= 0){
    fprintf (stderr, "%u:analyser - pthread_mutex_destroy mutex_analyser_thread_signalsender_error error: %s.\n", anal
  }

  /*
     Closing /proc/PID/stat
  */

  if (proc_process_statf != NULL){
    if ((fclose (proc_process_statf))==EOF){
      fprintf(stderr, "analyser - can't fclose '%s': %s\n", proc_process_stat, strerror(errno));
    }
  }

  if (analyseproc->flag & READFSWAPL){
    if ((mutex_error = pthread_mutex_lock(&(analyseproc->mutex_ffanalyser_thread_end))) != 0){
      fprintf (stderr, "analyser - pthread_mutex_lock error: %s.\n", strerror (mutex_error));
    }
    analyseproc->ffanalyser_thread_end = TRUE;
    if ((mutex_error = pthread_mutex_unlock(&(analyseproc->mutex_ffanalyser_thread_end))) != 0){
      fprintf (stderr, "analyser - pthread_mutex_unlock error: %s.\n", strerror (mutex_error));
    }
  }

  return status;
}
1295
static void
usage(FILE *ofp, int exitval)
{
  fprintf(ofp, "usage: panalyser [-d] [-f] [-h] [-e] [-i interval] [-u interval] [-t seconds] [-T minutes] [-s [-c ord
  fprintf(ofp, "-d -- print debug messages to stderr\n");
  fprintf(ofp, "-f interval_fforks -- follow forks of process with the specified interval between number of forks, cre
  fprintf(ofp, "-h -- print this help message\n");
  fprintf(ofp, "-e -- analyse the process until it ends. [-t -T] are ignored\n");
  fprintf(ofp, "-i seconds -- seconds between measurements\n");
  fprintf(ofp, "-u useconds -- microseconds between measurements\n");
  fprintf(ofp, "-t seconds -- duration of measurement in seconds\n");
  fprintf(ofp, "-T minutes -- duration of measurement in minutes\n");
  fprintf(ofp, "-s -- enable statistics such as mean, variance, std deviation, median and coef. of variation, this opt
  fprintf(ofp, "-c order criteria -- for median, can be one of: NSYSCALLS, UTIME, STIME\(default)\n");
  fprintf(ofp, "-m file -- temporary FILE name\n");
  fprintf(ofp, "-o file -- send trace output to FILE instead of stderr\n");
  fprintf(ofp, "-p pid -- trace process with process id PID\n");

  _exit(exitval);
```

```
}


int
main (int argc, char *argv[])
{
  /* options variables*/
  static char buf[BUFSIZ];
  char *outfname = NULL;
  FILE *outf;
  char *stfname_rusage = NULL;
  FILE *stf_rusage = NULL;
  char *stfname_syscall = NULL;
  FILE *stf_syscall = NULL;
  char *swap_outfname = NULL;
  char tmpstring[MAXPATHLEN];
  unsigned int c, seconds;
1332  unsigned long int tempo_medicao, tempo_medicao_min, useconds, tam_loop;
  char *progname;
  int debug = 0;
  unsigned short int criterio_ord = STIME;

  /* control variables */
  pid_t pid = 0;
  int status = EXIT_SUCCESS;
  int status_real = EXIT_SUCCESS;
  unsigned short int flag = 0;
  unsigned short int flag_swap = 0;
  lista_db lista, registro_apontador, registro_fim;
  char string_pid[N_PROC_DIGIT];
  unsigned int l;
  pointer_thread_ffork analyseproc = NULL;
  unsigned int interval_fforks = 0;

  outf = stdout;
  seconds = 0;
  useconds = 0;
  tempo_medicao = 0;
  tempo_medicao_min = 0;
  progname = argv[0];

  if (argc < 2){
    status = 1;
    usage(stderr, status);
    return status;
  }

  while ((c = getopt(argc, argv, "+hdf:sc:ei:u:t:T:m:o:p:")) != EOF) {
    switch (c) {
    case 'h':
      status = 0;
      usage(stdout, status);
      break;
    case 'd':
1369    debug++;
      break;
    case 'f':
      flag |= OPTFFORK;
      interval_fforks = (unsigned int) atol (optarg);
      if (interval_fforks <= 0){
        interval_fforks = 1;
      }
      break;
    case 's':
      flag |= STATISTICS;
      break;
    case 'c':
      if (flag & STATISTICS){
        if (strcmp(optarg, "NSYSCALLS") == 0){
          criterio_ord = NSYSCALLS;
        }
        else if (strcmp(optarg, "UTIME") == 0){
          criterio_ord = UTIME;
```

```
        }
        else if (strcmp(optarg, "STIME") == 0){
          criterio_ord = STIME;
        }
        else{
          status = 1;
          usage(stderr, status);
          break;
        }
      }else{
        status = 1;
        usage(stderr, status);
        break;
      }
      break;
    case 'e':
      flag |= UNTILEND;
      break;
1406  case 'i':
      seconds = (unsigned int) atol (optarg);
      break;
    case 'u':
      useconds = (unsigned long int) atol (optarg);
      break;
    case 't':
      tempo_medicao = (unsigned long int) atol (optarg);
      break;
    case 'T':
      tempo_medicao_min = (unsigned long int) atol (optarg);
      break;
    case 'm':
      swap_outfname = strdup(optarg);
      break;
    case 'o':
      outfname = strdup(optarg);
      break;
    case 'p':
      if ((pid = (pid_t) atoi (optarg)) == 0) {
        fprintf(stderr, "%s: Invalid process id: %s\n", progname, optarg);
        break;
      }
      if (pid == getpid()) {
        fprintf(stderr, "%s: It's invalid to trace the %s\n", progname, progname);
        break;
      }
      (flag |= ATTACHED);
      if (debug){
        fprintf (stderr, "\n%u:main\n", pid);
      }
      break;
    default:
      status = 1;
      usage(stderr, status);
      break;
    }
1443  }

  if (debug)
    fprintf (stderr, "\nmain - registro:%u\n", sizeof (registro));

  if (tempo_medicao == 0 && tempo_medicao_min == 0){
    tempo_medicao = DEFAULTTIME;
  }

  if (useconds > 0 && seconds > 0){
    tam_loop = (unsigned long int) (tempo_medicao + tempo_medicao_min * 60) / (seconds + ((double)useconds) / M);
  }else if (useconds > 0 && seconds == 0){
    tam_loop = (unsigned long int) (tempo_medicao + tempo_medicao_min * 60) / (((double)useconds) / M);
  }else if (useconds == 0 && seconds > 0){
    tam_loop = (unsigned long int) (tempo_medicao + tempo_medicao_min * 60) / seconds;
  } else {
    seconds = DEFAULTSLEEP;
    tam_loop = (unsigned long int) (tempo_medicao + tempo_medicao_min * 60) / seconds;
```

```
}

if (debug)
  fprintf (stderr, "\nmain - tam_loop:%lu\n", tam_loop);

/* See if they want to pipe the output. */
if (outfname && (outfname[0] == '|' || outfname[0] == '!')){
  if ((outf = popen(outfname + 1, "w")) == NULL) {
    fprintf(stderr, "%s: can't popen '%s': %s\n",
            progname, outfname + 1, strerror(errno));
    status = 1;
    _exit (status);
  }
  free(outfname);
  outfname = NULL;
}

/* Check if they want to redirect the output. */
if (outfname) {
1480  if ((outf = fopen(outfname, "w")) == NULL){
    fprintf(stderr, "%s:main - can't fopen '%s': %s\n", progname, outfname, strerror(errno));
    status = 1;
    _exit (status);
  }

  if (flag & STATISTICS){
    memset(tmpstring, '\0', sizeof (tmpstring));
    if (strlen(outfname) < (MAXPATHLEN - strlen(".rusage.dat"))){
      memcpy(tmpstring, outfname, strlen(outfname));
    } else {
      perror ("main - memcpy error");
      status = 1;
      _exit (status);
    }
    stfname_rusage = strdup(strcat (tmpstring, ".rusage.dat"));
    if ((stf_rusage = fopen(stfname_rusage, "w")) == NULL){
      fprintf(stderr, "%s:main - can't fopen '%s': %s\n", progname, stfname_rusage, strerror(errno));
      status = 1;
      _exit (status);
    }

    memset(tmpstring, '\0', sizeof (tmpstring));
    if (strlen(outfname) < (MAXPATHLEN - strlen(".syscall.dat"))){
      memcpy(tmpstring, outfname, strlen(outfname));
    } else {
      perror ("main - memcpy error");
      status = 1;
      _exit (status);
    }
    stfname_syscall = strdup(strcat (tmpstring, ".syscall.dat"));
    if ((stf_syscall = fopen(stfname_syscall, "w")) == NULL){
      fprintf(stderr, "%s:main - can't fopen '%s': %s\n", progname, stfname_syscall, strerror(errno));
      status = 1;
      _exit (status);
    }
  } /* if (flag & STATISTICS) */
1517 } else {
  if (flag & STATISTICS){
    stfname_rusage = strdup("proc_res_consump.rusage.dat");
    if ((stf_rusage = fopen(stfname_rusage, "w")) == NULL){
      fprintf(stderr, "%s:main - can't fopen '%s': %s\n", progname, stfname_rusage, strerror(errno));
      status = 1;
      _exit (status);
    }
    stfname_syscall = strdup("proc_res_consump.syscall.dat");
    if ((stf_syscall = fopen(stfname_syscall, "w")) == NULL){
      fprintf(stderr, "%s:main - can't fopen '%s': %s\n", progname, stfname_syscall, strerror(errno));
      status = 1;
      _exit (status);
    }
  } /* if (flag & STATISTICS) */
}
```

```
if (!outfname) {
  setvbuf(outf, buf, _IOLBF, BUFSIZ);
}

if ((optind < argc) && !(flag & ATTACHED)) {
  struct stat statbuf;
  char *filename;
  char pathname[MAXPATHLEN];

  filename = argv[optind];
  if (strchr(filename, '/'))
    strcpy(pathname, filename);
  else {
    char *path;
    int m, n, len;

    for (path = getenv("PATH"); path && *path; path += m) {
      if (strchr(path, ':')) {
        n = strchr(path, ':') - path;
        m = n + 1;
1554    }
      else
        m = n = strlen(path);
      if (n == 0) {
        getcwd(pathname, MAXPATHLEN);
        len = strlen(pathname);
      }
      else {
        strncpy(pathname, path, n);
        len = n;
      }
      if (len && pathname[len - 1] != '/')
        pathname[len++] = '/';
      strcpy(pathname + len, filename);
      if (stat(pathname, &statbuf) == 0)
        break;
    }
  }
  if (stat(pathname, &statbuf) < 0) {
    fprintf(stderr, "%s: %s: command not found\n",
            progname, filename);
    status = 1;
    _exit(status);
  }
  pid = fork();
  if (pid == 0){
    if (ptrace (PTRACE_TRACEME, 0, (char *) 1, 0) < 0) {
      fprintf (stderr, "main - ptrace (PTRACE_TRACEME, %u,... ) error: %s.\n", pid, strerror (errno));
      status = 1;
      _exit(status);
    }
    execv(pathname, &argv[optind]);
    perror("main - exec error");
    _exit(1);
  }
  else if (pid < 0 ){
    perror("main - fork error");
1591  status = 1;
    _exit(status);
  }
  else {
    flag |= WAITEXECVE;
  }
}

if ((analyseproc = (pointer_thread_ffork) malloc (sizeof(element_thread_ffork)))!=NULL){
  analyseproc->ffeax_number = 0;
  for (l = 0; l < MAXFORKSTOFOLLOW; analyseproc->ffeaxvector[l++] = 0);
  analyseproc->pid = pid;
  analyseproc->flag = flag;
  analyseproc->flag_swap = flag_swap;
  analyseproc->seconds = seconds;
  analyseproc->useconds = useconds;
```

```
  analyseproc->tam_loop = tam_loop;
  analyseproc->outf = outf;
  if (outfname){
    analyseproc->outfname = strdup (outfname);
  } else {
    analyseproc->outfname = NULL;
  }
  if (flag & STATISTICS) {
    analyseproc->stf_rusage = stf_rusage;
    analyseproc->stfname_rusage = strdup (stfname_rusage);
    analyseproc->stf_syscall = stf_syscall;
    analyseproc->stfname_syscall = strdup (stfname_syscall);
  }
  if (swap_outfname){
    analyseproc->swap_outfname = strdup(swap_outfname);
  } else {
    analyseproc->swap_outfname = NULL;
  }
  analyseproc->criterio_ord = criterio_ord;
  analyseproc->process_start_time = 0;
  analyseproc->panalyser_start_time = 0;
1628
  analyseproc->real_run_time = 0;
  analyseproc->effective_run_time.tv_sec = 0;
  analyseproc->effective_run_time.tv_usec = 0;
  for (l = 0; l < MAXFORKSTOFOLLOW; l++){
    analyseproc->ffanalyseproc[l] = NULL;
  }
  analyseproc->interval_fforks = interval_fforks;
  analyseproc->debug = debug;

} else {
  perror ("main - malloc of analyseproc error");
  status = 1;
  _exit(status);
}

if (analyser (analyseproc) == NULL){
  /*
    If the analyser thread return error don't interfere with the
    analysed process.
  */
  if (kill (pid, SIGCONT)<0){
    perror ("main - kill (pid,SIGCONT) error");
    status = 1;
  }
  perror ("main - analyser error");
  status = 1;
  _exit (status);
}

/* Saving results session */

init_list (&lista);

if (data_swap_out (&lista, pid, swap_outfname, debug) < 0){
  perror ("main - data_swap_out error");
  status = 1;
}
1665
if (status == 0){
  fprintf (outf, "\npid:%u\n", pid);
  fprintf (outf, "Exec parameters summary\n");
  fprintf (outf, "seconds between measurements:%ds\n", seconds);
  fprintf (outf, "microseconds between measurements:%ldus\n", useconds);
  registro_apontador = lista;
  registro_fim = localizar_fim (registro_apontador);
  fprintf (outf, "total measurement time:%1.10fs\n", registro_fim->momento_insercao);
  fprintf (outf, "total number of measurements:%lu\n", registro_fim->indice + 1);

  memset(string_pid, '\0', sizeof (string_pid));
  sprintf(string_pid, ".%u", (unsigned int) pid);
  if (swap_outfname){
    fprintf (outf, "temporary file location:%s%s\n", swap_outfname, string_pid);
```

```
  }
  else {
    fprintf (outf, "temporary file location:data_swap%s\n", string_pid);
  }
  fprintf (outf, "\nBegin process analysis ");
  print_proc_stat_summary (lista->proc_stat, outf, debug);
  fprintf (outf, "\nEnd process analysis ");
  print_proc_stat_summary (registro_fim->proc_stat, outf, debug);
  fprintf (outf, "Process start time:%s", ctime (&(analyseproc->process_start_time)));
  fprintf (outf, "Panalyser start time:%s", ctime (&(analyseproc->panalyser_start_time)));
  fprintf (outf, "Real time running:%1.10fs\n", analyseproc->real_run_time);
  fprintf (outf, "CPU percentage:%1.10f\n", ((analyseproc->effective_run_time.tv_sec + ((double) analyseproc->effect

  if (calc_present (lista, flag, criterio_ord, outf, stf_rusage, stf_syscall, debug) < 0){
    perror ("main - calc_present error");
    status = 1;
  }

  if (del_list (&lista) < 0){
    perror ("main - del_list error");
    status = 1;
  }
1702

  /* close output files */
  if ((fclose (outf))==EOF){
    perror ("main - fclose error");
    status = 1;
  }
  if (flag & STATISTICS) {
    if ((fclose (stf_rusage))==EOF){
      perror ("main - fclose error");
      status = 1;
    }
    if ((fclose (stf_syscall))==EOF){
      perror ("main - fclose error");
      status = 1;
    }
  }
} /* if (status == 0) */
status_real = status;
status = 0;

/*
  Repeat the above process for forked process from analysed process
*/

for (l = 0; l < analyseproc->ffeax_number; l++){
  if (analyseproc->ffeaxvector[l] > 0){
    lista_db ff_lista;
    FILE *ff_outf = NULL;
    char *ff_outfname = NULL;
    FILE *ff_stf_rusage = NULL;
    char *ff_stfname_rusage=NULL;
    FILE *ff_stf_syscall = NULL;
    char *ff_stfname_syscall=NULL;
    char tmpstring[MAXPATHLEN];
    char string_pid[N_PROC_DIGIT];


1739    init_list (&ff_lista);
    if (data_swap_out (&ff_lista, analyseproc->ffeaxvector[l], swap_outfname, debug) < 0){
      perror ("main - data_swap_out error");
      status = 1;
    }

    if (status == 0){
      /* files opening */
      memset(string_pid, '\0', sizeof (string_pid));
      sprintf(string_pid, ".%u", (unsigned int) analyseproc->ffeaxvector[l]);

      memset(tmpstring, '\0', sizeof (tmpstring));
      if (outfname) {
        if (strlen(outfname) < (MAXPATHLEN - strlen(string_pid))) {
```

```
      memcpy(tmpstring, outfname, strlen(outfname));
    } else {
      perror ("main - memcpy error");
      status = 1;
    }
    ff_outfname = strdup(strcat (tmpstring, string_pid));
    if ((ff_outf = fopen(ff_outfname, "w")) == NULL){
      fprintf(stderr, "main - can't fopen '%s': %s\n", ff_outfname, strerror(errno));
      status = 1;
    }
  }
  else {
    if  (strlen("proc_res_consump.log") < (MAXPATHLEN - strlen(string_pid))){
      memcpy(tmpstring,"proc_res_consump.log", strlen("proc_res_consump.log"));
    } else {
      perror ("main - memcpy error");
      status = 1;
    }
    ff_outfname = strdup(strcat (tmpstring, string_pid));
    if ((ff_outf = fopen(ff_outfname, "w")) == NULL){
      fprintf(stderr, "main - can't fopen '%s': %s\n", ff_outfname, strerror(errno));
      status = 1;
    }
1776  }

  if (flag & STATISTICS){
    memset(tmpstring, '\0', sizeof (tmpstring));
    if (strlen(stfname_rusage) < (MAXPATHLEN - strlen(string_pid))){
      memcpy(tmpstring, stfname_rusage, strlen(stfname_rusage));
    } else {
      perror ("main - memcpy error");
      status = 1;
    }
    ff_stfname_rusage = strdup(strcat (tmpstring, string_pid));
    if ((ff_stf_rusage = fopen(ff_stfname_rusage, "w")) == NULL){
      fprintf(stderr, "main - can't fopen '%s': %s\n", ff_stfname_rusage, strerror(errno));
      status = 1;
    }

    memset(tmpstring, '\0', sizeof (tmpstring));
    if (strlen(stfname_syscall) < (MAXPATHLEN - strlen(string_pid))){
      memcpy(tmpstring, stfname_syscall, strlen(stfname_syscall));
    } else {
      perror ("main - memcpy error");
      status = 1;
    }
    ff_stfname_syscall = strdup(strcat (tmpstring, string_pid));
    if ((ff_stf_syscall = fopen(ff_stfname_syscall, "w")) == NULL){
      fprintf(stderr, "main - can't fopen '%s': %s\n", ff_stfname_syscall, strerror(errno));
      status = 1;
    }
  } /* if (flag & STATISTICS) */
  /* End of files opening */
  if (status == 0){
    fprintf (ff_outf, "\npid:%u\n", analyseproc->ffeaxvector[1]);
    fprintf (ff_outf, "Exec parameters summary\n");
    fprintf (ff_outf, "seconds between measurements:%ds\n", seconds);
    fprintf (ff_outf, "microseconds between measurements:%ldus\n", useconds);
    registro_apontador = ff_lista;
    registro_fim = localizar_fim (registro_apontador);
1813  fprintf (ff_outf, "total measurement time:%1.10fs\n", registro_fim->momento_insercao);
    fprintf (ff_outf, "total number of measurements:%lu\n", registro_fim->indice + 1);
    if (swap_outfname){
      fprintf (ff_outf, "temporary file location:%s%s\n", swap_outfname, string_pid);
    }
    else {
      fprintf (ff_outf, "temporary file location:data_swap%s\n", string_pid);
    }
    fprintf (ff_outf, "\nBegin process analysis ");
    print_proc_stat_summary (ff_lista->proc_stat, ff_outf, debug);
    fprintf (ff_outf, "\nEnd process analysis ");
    print_proc_stat_summary (registro_fim->proc_stat, ff_outf, debug);
    fprintf (ff_outf, "Process start time:%s", ctime (&(analyseproc->ffanalyseproc[1]->process_start_time)));
```

```
            fprintf (ff_outf, "Panalyser start time:%s", ctime (&(analyseproc->ffanalyseproc[l]->panalyser_start_time)))
            fprintf (ff_outf, "Real time running:%1.10fs\n", analyseproc->ffanalyseproc[l]->real_run_time);
            fprintf (ff_outf, "CPU percentage:%1.10f\n", ((analyseproc->ffanalyseproc[l]->effective_run_time.tv_sec + ((

            if (calc_present (ff_lista, flag, criterio_ord, ff_outf, ff_stf_rusage, ff_stf_syscall, debug) < 0){
              perror ("main - calc_present error");
              status = 1;
            }

            if (del_list (&ff_lista) < 0){
              perror ("main - del_list error");
              status = 1;
            }

            /* close output files */
            if ((fclose (ff_outf))==EOF){
              perror ("main - fclose error");
              status = 1;
            }
            if (flag & STATISTICS) {
              if ((fclose (ff_stf_rusage))==EOF){
                perror ("main - fclose error");
                status = 1;
              }
              if ((fclose (ff_stf_syscall))==EOF){
                perror ("main - fclose error");
                status = 1;
              }
            }
            free (ff_outfname);
            free (ff_stfname_rusage);
            free (ff_stfname_syscall);
          } /* if (status == 0) */
          status_real = status;
          status = 0;
        } /* if (status == 0) */
        status_real = status;
        status = 0;
      } /* if (analyseproc->ffeaxvector[l] > 0) */
    } /* for (l = 0; l < analyseproc->ffeax_number; l++) */

    exit (status_real);
  }
```

```
1   /*
     * Copyright (c) 2001-2002 Martin Alain Kretschek <martink@osite.com.br>
     * All rights reserved.
     *
     * Redistribution and use in source and binary forms, with or without
     * modification, are permitted provided that the following conditions
     * are met:
     * 1. Redistributions of source code must retain the above copyright
     *    notice, this list of conditions and the following disclaimer.
     * 2. Redistributions in binary form must reproduce the above copyright
     *    notice, this list of conditions and the following disclaimer in the
     *    documentation and/or other materials provided with the distribution.
     * 3. The name of the author may not be used to endorse or promote products
     *    derived from this software without specific prior written permission.
     *
     * THIS SOFTWARE IS PROVIDED BY THE AUTHOR ``AS IS'' AND ANY EXPRESS OR
     * IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
     * OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED.
     * IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT,
     * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
     * NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
     * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
     * THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
     * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF
     * THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
     *
     *      $Id: list.c,v 1.0Release 2001/11/27 17:35:00 martin Exp $
     */

    #include "definitions.h"
    #include <math.h>

    int data_swap_in (registro swap_vector[], pid_t pid, char *user_swap_outfname, unsigned int elements_num, unsigned sho
         /*
            Writes data on swap file.
         */
37  {
      int status = 0;
      FILE *swap_outf;
      char *swap_outfname=NULL;
      char tmpstring[MAXPATHLEN];
      char string_pid[N_PROC_DIGIT];

      memset(string_pid, '\0', sizeof (string_pid));
      sprintf(string_pid, ".%u", (unsigned int) pid);

      if (debug)
        fprintf (stderr, "\n%u:data_swap_in\n", (unsigned int) pid);

      if (user_swap_outfname){
        memset(tmpstring, '\0', sizeof (tmpstring));
        if (strlen(user_swap_outfname) < (MAXPATHLEN - strlen(string_pid))){
          memcpy(tmpstring, user_swap_outfname, strlen(user_swap_outfname));
        } else {
          perror ("data_swap_in ~ memcpy error");
          status = -1;
          return status;
        }
        swap_outfname = strdup(strcat (tmpstring, string_pid));
        if ((swap_outf = fopen(swap_outfname, ((flag_swap & FIRSTSWAP) ? "w" : "a"))) == NULL){
          fprintf(stderr, "can't fopen '%s': %s\n", swap_outfname, strerror(errno));
          status = -1;
          return status;
        }

      }
      else{
        memset(tmpstring, '\0', sizeof (tmpstring));
        if (strlen("data_swap") < (MAXPATHLEN - strlen(string_pid))){
          memcpy(tmpstring, "data_swap", strlen("data_swap"));
        } else {
          perror ("data_swap_in - memcpy error");
          status = -1;
```

```
74        return status;
        }
        swap_outfname = strdup(strcat (tmpstring, string_pid));
        if ((swap_outf = fopen(swap_outfname, ((flag_swap & FIRSTSWAP) ? "w" : "a"))) == NULL){
          fprintf(stderr, "can't fopen '%s': %s\n", swap_outfname, strerror(errno));
          status = -1;
          return status;
        }

      }

      if (fwrite (swap_vector, sizeof(registro), elements_num, swap_outf) < elements_num){
        perror ("data_swap_in - fwrite, number of elements writen less the passed to function");
        status = -1;
        return status;
      }

      /* close output files */
      if ((fclose (swap_outf))==EOF){
        perror ("data_swap_in - fclose error");
        status = -1;
        return status;
      }

      return status;

    }


    int data_swap_out (lista_db *lista, pid_t pid, char *user_swap_infname, int debug)
         /*
            Read data from swap file.
         */
    {
      registro swap_register;
      int status = 0;
      FILE *swap_inf;
111   char *swap_infname=NULL;
      char tmpstring[MAXPATHLEN];
      char string_pid[N_PROC_DIGIT];
      unsigned long int indice = 0;

      memset(string_pid, '\0', sizeof (string_pid));
      sprintf(string_pid, ".%u", (unsigned int) pid);

      if (debug)
        fprintf (stderr, "\n%u:data_swap_out\n", (unsigned int) pid);

      if (user_swap_infname){
        memset(tmpstring, '\0', sizeof (tmpstring));
        if (strlen(user_swap_infname) < (MAXPATHLEN - strlen(string_pid))){
          memcpy(tmpstring, user_swap_infname, strlen(user_swap_infname));
        } else {
          perror ("data_swap_out - memcpy error");
          status = -1;
          return status;
        }
        swap_infname = strdup(strcat (tmpstring, string_pid));
        if ((swap_inf = fopen(swap_infname, "r")) == NULL){
          fprintf(stderr, "can't fopen '%s': %s\n", swap_infname, strerror(errno));
          status = -1;
          return status;
        }

      }
      else{
        memset(tmpstring, '\0', sizeof (tmpstring));
        if (strlen("data_swap") < (MAXPATHLEN - strlen(string_pid))){
          memcpy(tmpstring, "data_swap", strlen("data_swap"));
        } else {
          perror ("data_swap_out - memcpy error");
          status = -1;
          return status;
```

```
    }
148 swap_infname = strdup(strcat (tmpstring, string_pid));
    if ((swap_inf = fopen(swap_infname, "r")) == NULL){
      fprintf(stderr, "can't fopen '%s': %s\n", swap_infname, strerror(errno));
      status = -1;
      return status;
    }
  }
  free (swap_infname);

  clearerr (swap_inf);
  while (feof (swap_inf) == 0){

    fread (&swap_register, sizeof(registro), 1, swap_inf);
    if (ferror (swap_inf)){
      perror ("data_swap_out - fread error");
      status = -1;
      return status;
    }

    if ((indice < swap_register.indice) || (swap_register.indice == 0)){
      if (insert_end (lista, swap_register.indice, swap_register.momento_insercao, swap_register.sys_res_usage, swap_r
        perror ("data_swap_out - insert_end error");
        status = -1;
        return status;
      }
    }
    indice = swap_register.indice;
  }

  /* close output files */
  if ((fclose (swap_inf))==EOF){
    perror ("data_swap_out - fclose error");
    status = -1;
    return status;
  }

  return status;
185
}


int init_list (lista_db * lista)
{

  *lista = NULL;
  return 0;
}


int insert_end (lista_db * lista, unsigned long int indice, double momento_insercao, struct rusage sys_res_usage, elem
     /*
        Inserts a new node on the list.
     */
{

  lista_db registro_novo, registro_apontador, registro_fim;
  unsigned int i;

  if ((registro_novo = (lista_db) malloc (sizeof(registro)))!=NULL){
    registro_novo->indice = indice;
    registro_novo->momento_insercao = momento_insercao;
    registro_novo->sys_res_usage = sys_res_usage;
    registro_novo->proc_stat = proc_stat;
    registro_novo->sys_call_in_cont = sys_call_in_cont;
    for (i=0;i<NO_SYS_CALLS_PLUSONE;i++){
      registro_novo->sys_call_no[i] = sys_call_no[i];
    }
    registro_novo->prox = NULL;
    registro_novo->ant = NULL;
  }
  else {
    perror ("insert_end - malloc error");
```

```
    return -1;
  }
222
  if (*lista != NULL){
    registro_apontador = *lista;
    registro_fim = localizar_fim (registro_apontador);
    registro_fim->prox = registro_novo;
    registro_novo->ant = registro_fim;

    if (timeval_subtract (&registro_novo->diff_ru_utime, registro_novo->sys_res_usage.ru_utime, registro_novo->ant->sy
      perror ("insert_end - timeval_subtract error");
      return -1;
    }
    if (timeval_subtract (&registro_novo->diff_ru_stime, registro_novo->sys_res_usage.ru_stime, registro_novo->ant->sy
      perror ("insert_end - timeval_subtract error");
      return -1;
    }
    registro_novo->diff_sys_call_in_cont = registro_novo->sys_call_in_cont - registro_novo->ant->sys_call_in_cont;

    if (debug){
      fprintf (stderr, "\ninsert_end - lista != NULL lista->indice:%lu\n", registro_fim->prox->indice);
      fprintf (stderr, "insert_end - diff_ru_utime:%.15f\n", (((double) registro_novo->diff_ru_utime.tv_sec) * M + reg
      fprintf (stderr, "insert_end - diff_ru_stime:%.15f\n", (((double) registro_novo->diff_ru_stime.tv_sec) * M + reg
      fprintf (stderr, "insert_end - diff_sys_call_in_cont:%.15f\n", ((double) registro_novo->diff_sys_call_in_cont));
    }

  }
  else{
    registro_novo->diff_ru_utime.tv_sec = 0;
    registro_novo->diff_ru_utime.tv_usec = 0;
    registro_novo->diff_ru_stime.tv_sec = 0;
    registro_novo->diff_ru_stime.tv_usec = 0;
    registro_novo->diff_sys_call_in_cont = 0;
    *lista = registro_novo;

    if (debug){
      fprintf (stderr, "\ninsert_end - lista = NULL lista->indice:%lu\n", (*lista)->indice);
      fprintf (stderr, "insert_end - diff_ru_utime:%.15f\n", (((double) registro_novo->diff_ru_utime.tv_sec) * M + reg
      fprintf (stderr, "insert_end - diff_ru_stime:%.15f\n", (((double) registro_novo->diff_ru_stime.tv_sec) * M + reg
259   fprintf (stderr, "insert_end - diff_sys_call_in_cont:%.15f\n", ((double) registro_novo->diff_sys_call_in_cont));
    }

  }

  return 0;

}


lista_db localizar_fim (lista_db lista)
     /*
       returns a pointer to the end of the list
     */
{
  while (lista->prox != NULL){
    lista = lista->prox;
  }

  return (lista);

}


int del_list (lista_db *lista)
     /*
       Delete list.
     */

{
  lista_db lista_inicio;

  if ((* lista)->prox != NULL){
    lista_inicio = (* lista)->prox;
```

```
    lista_inicio->ant = NULL;
    free (* lista);
    del_list (&lista_inicio);
296 }
  else {
    free (* lista);
  }
  return 0;
}


int
print_list (lista_db lista, unsigned short int flag, char *sys_call_name[], struct statistic_rusage media_sys_res_usag
{
  struct rusage diff_sys_res_usage_st;
  unsigned long int diff_sys_call_in_cont_st;
  unsigned long int diff_sys_call_no_st[NO_SYS_CALLS_PLUSONE];
  lista_db registro_apontador;
  unsigned int i;

  registro_apontador = lista;

  diff_sys_res_usage_st.ru_utime.tv_usec = 0;
  diff_sys_res_usage_st.ru_utime.tv_sec = 0;
  diff_sys_res_usage_st.ru_stime.tv_usec = 0;
  diff_sys_res_usage_st.ru_stime.tv_sec = 0;
  diff_sys_res_usage_st.ru_maxrss = 0;
  diff_sys_res_usage_st.ru_ixrss = 0;
  diff_sys_res_usage_st.ru_idrss = 0;
  diff_sys_res_usage_st.ru_isrss = 0;
  diff_sys_res_usage_st.ru_minflt = 0;
  diff_sys_res_usage_st.ru_majflt = 0;
  diff_sys_res_usage_st.ru_nswap = 0;
  diff_sys_res_usage_st.ru_inblock = 0;
  diff_sys_res_usage_st.ru_oublock = 0;
  diff_sys_res_usage_st.ru_msgsnd = 0;
  diff_sys_res_usage_st.ru_msgrcv = 0;
  diff_sys_res_usage_st.ru_nsignals = 0;
  diff_sys_res_usage_st.ru_nvcsw = 0;
  diff_sys_res_usage_st.ru_nivcsw = 0;
333

  diff_sys_call_in_cont_st = 0;

  for (i=0; i<NO_SYS_CALLS_PLUSONE; diff_sys_call_no_st[i++] = 0);

  while (registro_apontador != NULL){
    if (debug)
      fprintf (stderr, "\nprint_list - lista->indice:%lu\n", registro_apontador->indice);
    if (registro_apontador->ant == NULL) {
      fprintf (file, "\ncounter:%lu\ninsert time:%.15fs\n", registro_apontador->indice, registro_apontador->momento_ir
      print_getrusage_summary (registro_apontador->sys_res_usage, media_sys_res_usage_st, desviopadrao_sys_res_usage_s
      print_sys_call_summary (registro_apontador->sys_call_no, sys_call_name, media_sys_call_no_st, desviopadrao_sys_c
      fprintf (file, "\n");
    }
    else{
      /* Calculate the difference between current node and the previous node. */
      if (timeval_subtract (&diff_sys_res_usage_st.ru_utime, registro_apontador->sys_res_usage.ru_utime, registro_apor
        perror ("print_list - timeval_subtract error");
        return -1;
      }
      if (timeval_subtract (&diff_sys_res_usage_st.ru_stime, registro_apontador->sys_res_usage.ru_stime, registro_apor
        perror ("print_list - timeval_subtract error");
        return -1;
      }
      diff_sys_res_usage_st.ru_maxrss = registro_apontador->sys_res_usage.ru_maxrss - registro_apontador->ant->sys_res
      diff_sys_res_usage_st.ru_ixrss = registro_apontador->sys_res_usage.ru_ixrss - registro_apontador->ant->sys_res_u
      diff_sys_res_usage_st.ru_idrss = registro_apontador->sys_res_usage.ru_idrss - registro_apontador->ant->sys_res_u
      diff_sys_res_usage_st.ru_isrss = registro_apontador->sys_res_usage.ru_isrss - registro_apontador->ant->sys_res_u
      diff_sys_res_usage_st.ru_minflt = registro_apontador->sys_res_usage.ru_minflt - registro_apontador->ant->sys_res
      diff_sys_res_usage_st.ru_majflt = registro_apontador->sys_res_usage.ru_majflt - registro_apontador->ant->sys_res
      diff_sys_res_usage_st.ru_nswap = registro_apontador->sys_res_usage.ru_nswap - registro_apontador->ant->sys_res_u
      diff_sys_res_usage_st.ru_inblock = registro_apontador->sys_res_usage.ru_inblock - registro_apontador->ant->sys_r
      diff_sys_res_usage_st.ru_oublock = registro_apontador->sys_res_usage.ru_oublock - registro_apontador->ant->sys_r
```

```
    diff_sys_res_usage_st.ru_msgsnd = registro_apontador->sys_res_usage.ru_msgsnd - registro_apontador->ant->sys_res.
    diff_sys_res_usage_st.ru_msgrcv = registro_apontador->sys_res_usage.ru_msgrcv - registro_apontador->ant->sys_res.
    /*
      Subtract panalyser signals.
370 */
    (((diff_sys_res_usage_st.ru_nsignals = registro_apontador->sys_res_usage.ru_nsignals - registro_apontador->ant->:
    diff_sys_res_usage_st.ru_nvcsw = registro_apontador->sys_res_usage.ru_nvcsw - registro_apontador->ant->sys_res_u:
    diff_sys_res_usage_st.ru_nivcsw = registro_apontador->sys_res_usage.ru_nivcsw - registro_apontador->ant->sys_res.

    diff_sys_call_in_cont_st = registro_apontador->sys_call_in_cont - registro_apontador->ant->sys_call_in_cont;

    for (i=0;i<NO_SYS_CALLS_PLUSONE;i++){
      diff_sys_call_no_st[i] = registro_apontador->sys_call_no[i] - registro_apontador->ant->sys_call_no[i];
    }
    fprintf (file, "\ncounter:%lu\ninsert time:%.15fs\n", registro_apontador->indice, registro_apontador->momento_in:
    print_getrusage_summary (registro_apontador->sys_res_usage, media_sys_res_usage_st, desviopadrao_sys_res_usage_s
    print_sys_call_summary (registro_apontador->sys_call_no, sys_call_name, media_sys_call_no_st, desviopadrao_sys_c
    fprintf (file, "\n");
  }

  registro_apontador = registro_apontador->prox;
}

if (flag & STATISTICS){
  diff_sys_res_usage_st.ru_utime.tv_usec = 0;
  diff_sys_res_usage_st.ru_utime.tv_sec = 0;
  diff_sys_res_usage_st.ru_stime.tv_usec = 0;
  diff_sys_res_usage_st.ru_stime.tv_sec = 0;
  diff_sys_res_usage_st.ru_maxrss = 0;
  diff_sys_res_usage_st.ru_ixrss = 0;
  diff_sys_res_usage_st.ru_idrss = 0;
  diff_sys_res_usage_st.ru_isrss = 0;
  diff_sys_res_usage_st.ru_minflt = 0;
  diff_sys_res_usage_st.ru_majflt = 0;
  diff_sys_res_usage_st.ru_nswap = 0;
  diff_sys_res_usage_st.ru_inblock = 0;
  diff_sys_res_usage_st.ru_oublock = 0;
  diff_sys_res_usage_st.ru_msgsnd = 0;
  diff_sys_res_usage_st.ru_msgrcv = 0;
  diff_sys_res_usage_st.ru_nsignals = 0;
  diff_sys_res_usage_st.ru_nvcsw = 0;
407 diff_sys_res_usage_st.ru_nivcsw = 0;

  diff_sys_call_in_cont_st = 0;

  for (i=0; i<NO_SYS_CALLS_PLUSONE; diff_sys_call_no_st[i++] = 0);

  registro_apontador = lista;
  while (registro_apontador != NULL){
    if (debug)
      fprintf (stderr, "\nprint_list - lista->indice:%lu\n", registro_apontador->indice);
    if (registro_apontador->ant == NULL) {
      print_getrusage_summary_table (registro_apontador->indice, registro_apontador->momento_insercao, registro_apon
      fprintf (stfile_rusage, "\n");
    }
    else{
      /* Calculate the difference between current node and previous node. */
      if (timeval_subtract (&diff_sys_res_usage_st.ru_utime, registro_apontador->sys_res_usage.ru_utime, registro_ap
        perror ("print_list - timeval_subtract error");
        return -1;
      }
      if (timeval_subtract (&diff_sys_res_usage_st.ru_stime, registro_apontador->sys_res_usage.ru_stime, registro_ap
        perror ("print_list - timeval_subtract error");
        return -1;
      }
      diff_sys_res_usage_st.ru_maxrss = registro_apontador->sys_res_usage.ru_maxrss - registro_apontador->ant->sys_r
      diff_sys_res_usage_st.ru_ixrss = registro_apontador->sys_res_usage.ru_ixrss - registro_apontador->ant->sys_res.
      diff_sys_res_usage_st.ru_idrss = registro_apontador->sys_res_usage.ru_idrss - registro_apontador->ant->sys_res.
      diff_sys_res_usage_st.ru_isrss = registro_apontador->sys_res_usage.ru_isrss - registro_apontador->ant->sys_res.
      diff_sys_res_usage_st.ru_minflt = registro_apontador->sys_res_usage.ru_minflt - registro_apontador->ant->sys_r
      diff_sys_res_usage_st.ru_majflt = registro_apontador->sys_res_usage.ru_majflt - registro_apontador->ant->sys_r
      diff_sys_res_usage_st.ru_nswap = registro_apontador->sys_res_usage.ru_nswap - registro_apontador->ant->sys_res.
      diff_sys_res_usage_st.ru_inblock = registro_apontador->sys_res_usage.ru_inblock - registro_apontador->ant->sys.
```

```
        diff_sys_res_usage_st.ru_oublock = registro_apontador->sys_res_usage.ru_oublock - registro_apontador->ant->sys
        diff_sys_res_usage_st.ru_msgsnd = registro_apontador->sys_res_usage.ru_msgsnd - registro_apontador->ant->sys_r
        diff_sys_res_usage_st.ru_msgrcv = registro_apontador->sys_res_usage.ru_msgrcv - registro_apontador->ant->sys_r
        /*
          Subtract panalyser signals.
444     */
        (((diff_sys_res_usage_st.ru_nsignals = registro_apontador->sys_res_usage.ru_nsignals - registro_apontador->ant
        diff_sys_res_usage_st.ru_nvcsw = registro_apontador->sys_res_usage.ru_nvcsw - registro_apontador->ant->sys_res
        diff_sys_res_usage_st.ru_nivcsw = registro_apontador->sys_res_usage.ru_nivcsw - registro_apontador->ant->sys_r


        diff_sys_call_in_cont_st = registro_apontador->sys_call_in_cont - registro_apontador->ant->sys_call_in_cont;

        print_getrusage_summary_table (registro_apontador->indice, registro_apontador->momento_insercao, registro_apon
        fprintf (stfile_rusage, "\n");
      }

      registro_apontador = registro_apontador->prox;
    }

    registro_apontador = lista;
    while (registro_apontador != NULL){
      if (debug)
        fprintf (stderr, "\nprint_list - lista->indice:%lu\n", registro_apontador->indice);
      if (registro_apontador->ant == NULL) {
        print_sys_call_summary_table (registro_apontador->indice, registro_apontador->momento_insercao, registro_apont
        fprintf (stfile_syscall, "\n");
      }
      else{
        /* Calculate the difference between current node and previous node. */
        for (i=0;i<NO_SYS_CALLS_PLUSONE;i++){
          diff_sys_call_no_st[i] = registro_apontador->sys_call_no[i] - registro_apontador->ant->sys_call_no[i];
        }
        print_sys_call_summary_table (registro_apontador->indice, registro_apontador->momento_insercao, registro_apon
        fprintf (stfile_syscall, "\n");
      }

      registro_apontador = registro_apontador->prox;
    }
  } /* if (flag & STATISTICS) */

  return 0;
481 }


int timeval_subtract (struct timeval *result, struct timeval x, struct timeval y, int debug)
     /*
        Subtract the 'struct timeval' values X and Y,
        storing the result in RESULT.
        Return 1 if the difference is negative, otherwise 0.
     */
{
  /* Perform the carry for the later subtraction by updating Y. */
  if (x.tv_usec < y.tv_usec) {
    int nsec = (y.tv_usec - x.tv_usec) / M + 1;
    y.tv_usec -= M * nsec;
    y.tv_sec += nsec;
  }
  if (x.tv_usec - y.tv_usec > M) {
    int nsec = (x.tv_usec - y.tv_usec) / M;
    y.tv_usec += M * nsec;
    y.tv_sec -= nsec;
  }

  /*
     Compute the time remaining to wait.
     'tv_usec' is certainly positive.
  */
  result->tv_sec = x.tv_sec - y.tv_sec;
  result->tv_usec = x.tv_usec - y.tv_usec;

  /* Return 1 if result is negative. */
  return (x.tv_sec < y.tv_sec);
```

```
}


int timeval_add (struct timeval *result, struct timeval x, struct timeval y, int debug)
     /*
       Add the 'struct timeval' values X and Y,
518    storing the result in RESULT.
     */
{
  /* Perform the carry for the later add by updating X. */
  if (x.tv_usec + y.tv_usec > 999999) {
    int nsec = (x.tv_usec + y.tv_usec) / M;
    x.tv_usec -= M * nsec;
    x.tv_sec += nsec;
  }

  /* Compute the time */
  result->tv_sec = y.tv_sec + x.tv_sec;
  result->tv_usec = y.tv_usec + x.tv_usec;

  /* Return 1 if result is negative. Variable capacity is not enough. */
  return (result->tv_sec < 0);
}


int timeval_div (struct timeval *result, struct timeval x, double y, int debug)
     /*
        Computes de value 'struct timeval' values X / Y,
        storing the result in RESULT.
     */
{
  double div_result;

  /* Compute the time. */
  div_result = (x.tv_sec + ((double) (x.tv_usec)) / M) / y;

  if (debug)
    fprintf (stderr, "\ntimeval_div - div_result:%.15f\n",div_result);

  result->tv_sec = (long int) div_result;
  result->tv_usec = (long int) ((div_result * M) - (((long int) div_result) * M));

  /* Return 1 if result is negative. Variable capacity is not enough. */
555  return (result->tv_sec < 0);
}



int timeval_pow (struct timeval *result, struct timeval x, double y, int debug)
     /*
        Raises 'struct timeval' values X to the power of Y,
        storing the result in RESULT.
     */
{
  double pow_result;

  /* Compute the time */
  pow_result = pow ((x.tv_sec + ((double) x.tv_usec) / M), y);

  if (debug)
    fprintf (stderr, "\ntimeval_pow - pow_result:%.15f\n",pow_result);

  result->tv_sec = (long int) pow_result;
  result->tv_usec = (long int) ((pow_result * M) - (((long int) pow_result) * M));

  /* Return 1 if result is negative. Variable capacity is not enough. */
  return (result->tv_sec < 0);
}


int timeval_sqrt (struct timeval *result, struct timeval x, int debug)
     /*
        Returns square root 'struct timeval' values of X ,
```

```
        storing the result in RESULT.
     */
{
  double sqrt_result;

  /* Compute the time */
  sqrt_result = sqrt (x.tv_sec + ((double) x.tv_usec) / M);
592
  if (debug)
    fprintf (stderr, "\ntimeval_sqrt - sqrt_result:%.15f\n",sqrt_result);

  result->tv_sec = (long int) sqrt_result;
  result->tv_usec = (long int) ((sqrt_result * M) - (((long int) sqrt_result) * M));

  /* Return 1 if result is negative. Variable capacity is not enough. */
  return (result->tv_sec < 0);
}


int mean_calc (lista_db lista, struct statistic_rusage *sys_res_usage_st, double *cpu_percentage_st, double *sys_call_
{
  struct rusage diff_sys_res_usage_st, ac_sys_res_usage_st;
  double ac_cpu_percentage_st;
  unsigned long int diff_sys_call_in_cont_st, ac_sys_call_in_cont_st;
  unsigned int i;
  unsigned long int divisor;
  unsigned long int diff_sys_call_no_st[NO_SYS_CALLS_PLUSONE], ac_sys_call_no_st[NO_SYS_CALLS_PLUSONE];
  lista_db registro_apontador;

  registro_apontador = lista;

  ac_sys_res_usage_st.ru_utime.tv_usec = 0;
  ac_sys_res_usage_st.ru_utime.tv_sec = 0;
  ac_sys_res_usage_st.ru_stime.tv_usec = 0;
  ac_sys_res_usage_st.ru_stime.tv_sec = 0;
  ac_sys_res_usage_st.ru_maxrss = 0;
  ac_sys_res_usage_st.ru_ixrss = 0;
  ac_sys_res_usage_st.ru_idrss = 0;
  ac_sys_res_usage_st.ru_isrss = 0;
  ac_sys_res_usage_st.ru_minflt = 0;
  ac_sys_res_usage_st.ru_majflt = 0;
  ac_sys_res_usage_st.ru_nswap = 0;
  ac_sys_res_usage_st.ru_inblock = 0;
  ac_sys_res_usage_st.ru_oublock = 0;
629  ac_sys_res_usage_st.ru_msgsnd = 0;
  ac_sys_res_usage_st.ru_msgrcv = 0;
  ac_sys_res_usage_st.ru_nsignals = 0;
  ac_sys_res_usage_st.ru_nvcsw = 0;
  ac_sys_res_usage_st.ru_nivcsw = 0;

  ac_cpu_percentage_st = 0;
  ac_sys_call_in_cont_st = 0;

  for (i=0; i<NO_SYS_CALLS_PLUSONE; ac_sys_call_no_st[i++] = 0);

  while (registro_apontador->prox != NULL){
    /* Calculate the difference between current node and next node. */
    if (timeval_subtract (&diff_sys_res_usage_st.ru_utime, registro_apontador->prox->sys_res_usage.ru_utime, registro
      perror ("mean_calc - timeval_subtract error");
      return -1;
    }
    if (timeval_subtract (&diff_sys_res_usage_st.ru_stime, registro_apontador->prox->sys_res_usage.ru_stime, registro
      perror ("mean_calc - timeval_subtract error");
      return -1;
    }
    diff_sys_res_usage_st.ru_maxrss = registro_apontador->prox->sys_res_usage.ru_maxrss - registro_apontador->sys_res
    diff_sys_res_usage_st.ru_ixrss = registro_apontador->prox->sys_res_usage.ru_ixrss - registro_apontador->sys_res_u
    diff_sys_res_usage_st.ru_idrss = registro_apontador->prox->sys_res_usage.ru_idrss - registro_apontador->sys_res_u
    diff_sys_res_usage_st.ru_isrss = registro_apontador->prox->sys_res_usage.ru_isrss - registro_apontador->sys_res_u
    diff_sys_res_usage_st.ru_minflt = registro_apontador->prox->sys_res_usage.ru_minflt - registro_apontador->sys_res
    diff_sys_res_usage_st.ru_majflt = registro_apontador->prox->sys_res_usage.ru_majflt - registro_apontador->sys_res
    diff_sys_res_usage_st.ru_nswap = registro_apontador->prox->sys_res_usage.ru_nswap - registro_apontador->sys_res_u
    diff_sys_res_usage_st.ru_inblock = registro_apontador->prox->sys_res_usage.ru_inblock - registro_apontador->sys_r
```

```
  diff_sys_res_usage_st.ru_oublock = registro_apontador->prox->sys_res_usage.ru_oublock - registro_apontador->sys_re
  diff_sys_res_usage_st.ru_msgsnd = registro_apontador->prox->sys_res_usage.ru_msgsnd - registro_apontador->sys_res_
  diff_sys_res_usage_st.ru_msgrcv = registro_apontador->prox->sys_res_usage.ru_msgrcv - registro_apontador->sys_res_
  /*
    Subtract panalyser signals.
  */
  (((diff_sys_res_usage_st.ru_nsignals = registro_apontador->prox->sys_res_usage.ru_nsignals - registro_apontador->s
  diff_sys_res_usage_st.ru_nvcsw = registro_apontador->prox->sys_res_usage.ru_nvcsw - registro_apontador->sys_res_us
666 diff_sys_res_usage_st.ru_nivcsw = registro_apontador->prox->sys_res_usage.ru_nivcsw - registro_apontador->sys_res_

  diff_sys_call_in_cont_st = registro_apontador->prox->sys_call_in_cont - registro_apontador->sys_call_in_cont;

  for (i=0;i<NO_SYS_CALLS_PLUSONE;i++){
    diff_sys_call_no_st[i] = registro_apontador->prox->sys_call_no[i] - registro_apontador->sys_call_no[i];
  }

  /* Acumulate differences for mean computation. */
  if (timeval_add (&ac_sys_res_usage_st.ru_utime, ac_sys_res_usage_st.ru_utime, diff_sys_res_usage_st.ru_utime, debu
    perror ("mean_calc - timeval_add error");
    return -1;
  }
  if (timeval_add (&ac_sys_res_usage_st.ru_stime, ac_sys_res_usage_st.ru_stime, diff_sys_res_usage_st.ru_stime, debu
    perror ("mean_calc - timeval_add error");
    return -1;
  }
  ac_sys_res_usage_st.ru_maxrss = ac_sys_res_usage_st.ru_maxrss + diff_sys_res_usage_st.ru_maxrss;
  ac_sys_res_usage_st.ru_ixrss = ac_sys_res_usage_st.ru_ixrss + diff_sys_res_usage_st.ru_ixrss;
  ac_sys_res_usage_st.ru_idrss = ac_sys_res_usage_st.ru_idrss + diff_sys_res_usage_st.ru_idrss;
  ac_sys_res_usage_st.ru_isrss = ac_sys_res_usage_st.ru_isrss + diff_sys_res_usage_st.ru_isrss;
  ac_sys_res_usage_st.ru_minflt = ac_sys_res_usage_st.ru_minflt + diff_sys_res_usage_st.ru_minflt;
  ac_sys_res_usage_st.ru_majflt = ac_sys_res_usage_st.ru_majflt + diff_sys_res_usage_st.ru_majflt;
  ac_sys_res_usage_st.ru_nswap = ac_sys_res_usage_st.ru_nswap + diff_sys_res_usage_st.ru_nswap;
  ac_sys_res_usage_st.ru_inblock = ac_sys_res_usage_st.ru_inblock + diff_sys_res_usage_st.ru_inblock;
  ac_sys_res_usage_st.ru_oublock = ac_sys_res_usage_st.ru_oublock +  diff_sys_res_usage_st.ru_oublock;
  ac_sys_res_usage_st.ru_msgsnd = ac_sys_res_usage_st.ru_msgsnd + diff_sys_res_usage_st.ru_msgsnd;
  ac_sys_res_usage_st.ru_msgrcv = ac_sys_res_usage_st.ru_msgrcv + diff_sys_res_usage_st.ru_msgrcv;
  ac_sys_res_usage_st.ru_nsignals = ac_sys_res_usage_st.ru_nsignals + diff_sys_res_usage_st.ru_nsignals;
  ac_sys_res_usage_st.ru_nvcsw = ac_sys_res_usage_st.ru_nvcsw + diff_sys_res_usage_st.ru_nvcsw;
  ac_sys_res_usage_st.ru_nivcsw = ac_sys_res_usage_st.ru_nivcsw + diff_sys_res_usage_st.ru_nivcsw;

  ac_cpu_percentage_st = ac_cpu_percentage_st + registro_apontador->prox->proc_stat.cpu_percentage;

  ac_sys_call_in_cont_st = ac_sys_call_in_cont_st + diff_sys_call_in_cont_st;

  for (i=0;i<NO_SYS_CALLS_PLUSONE;i++){
703 ac_sys_call_no_st[i] = ac_sys_call_no_st[i] + diff_sys_call_no_st[i];
  }

  registro_apontador = registro_apontador->prox;
}

/* Calculate mean */
divisor = registro_apontador->indice;
if (timeval_div (&sys_res_usage_st->ru_utime_st, ac_sys_res_usage_st.ru_utime, divisor, debug) != 0){
  perror ("mean_calc - timeval_div error");
  return -1;
}
if (timeval_div (&sys_res_usage_st->ru_stime_st, ac_sys_res_usage_st.ru_stime, divisor, debug) != 0){
  perror ("mean_calc - timeval_div error");
  return -1;
}
sys_res_usage_st->ru_maxrss_st = ((double) ac_sys_res_usage_st.ru_maxrss) / divisor;
sys_res_usage_st->ru_ixrss_st = ((double) ac_sys_res_usage_st.ru_ixrss) / divisor;
sys_res_usage_st->ru_idrss_st = ((double) ac_sys_res_usage_st.ru_idrss) / divisor;
sys_res_usage_st->ru_isrss_st = ((double) ac_sys_res_usage_st.ru_isrss) / divisor;
sys_res_usage_st->ru_minflt_st = ((double) ac_sys_res_usage_st.ru_minflt) / divisor;
sys_res_usage_st->ru_majflt_st = ((double) ac_sys_res_usage_st.ru_majflt) / divisor;
sys_res_usage_st->ru_nswap_st = ((double) ac_sys_res_usage_st.ru_nswap) / divisor;
sys_res_usage_st->ru_inblock_st = ((double) ac_sys_res_usage_st.ru_inblock) / divisor;
sys_res_usage_st->ru_oublock_st = ((double) ac_sys_res_usage_st.ru_oublock) / divisor;
sys_res_usage_st->ru_msgsnd_st = ((double) ac_sys_res_usage_st.ru_msgsnd) / divisor;
sys_res_usage_st->ru_msgrcv_st = ((double) ac_sys_res_usage_st.ru_msgrcv) / divisor;
sys_res_usage_st->ru_nsignals_st = ((double) ac_sys_res_usage_st.ru_nsignals) / divisor;
```

```
  sys_res_usage_st->ru_nvcsw_st = ((double) ac_sys_res_usage_st.ru_nvcsw) / divisor;
  sys_res_usage_st->ru_nivcsw_st = ((double) ac_sys_res_usage_st.ru_nivcsw) / divisor;

  *cpu_percentage_st = (ac_cpu_percentage_st) / divisor;

  *sys_call_in_cont_st = ((double) ac_sys_call_in_cont_st) / divisor;

  for (i=0;i<NO_SYS_CALLS_PLUSONE;i++){
    sys_call_no_st[i] = ((double) ac_sys_call_no_st[i]) / divisor;
740 }

  return 0;
}


int variance_calc (lista_db lista, struct statistic_rusage *sys_res_usage_st, double *cpu_percentage_st, double *sys_c

{
  struct rusage diff_sys_res_usage_st;
  struct statistic_rusage ac_sys_res_usage_st, diffmedia_sys_res_usage_st, pow_sys_res_usage_st;
  double ac_cpu_percentage_st, diffmedia_cpu_percentage_st, pow_cpu_percentage_st;
  unsigned long int diff_sys_call_in_cont_st;
  double ac_sys_call_in_cont_st, diffmedia_sys_call_in_cont_st, pow_sys_call_in_cont_st;
  unsigned long int diff_sys_call_no_st[NO_SYS_CALLS_PLUSONE];
  double ac_sys_call_no_st[NO_SYS_CALLS_PLUSONE], diffmedia_sys_call_no_st[NO_SYS_CALLS_PLUSONE], pow_sys_call_no_st[N
  unsigned int i;
  unsigned long int divisor;
  lista_db registro_apontador;

  registro_apontador = lista;

  ac_sys_res_usage_st.ru_utime_st.tv_usec = 0;
  ac_sys_res_usage_st.ru_utime_st.tv_sec = 0;
  ac_sys_res_usage_st.ru_stime_st.tv_usec = 0;
  ac_sys_res_usage_st.ru_stime_st.tv_sec = 0;
  ac_sys_res_usage_st.ru_maxrss_st = 0;
  ac_sys_res_usage_st.ru_ixrss_st = 0;
  ac_sys_res_usage_st.ru_idrss_st = 0;
  ac_sys_res_usage_st.ru_isrss_st = 0;
  ac_sys_res_usage_st.ru_minflt_st = 0;
  ac_sys_res_usage_st.ru_majflt_st = 0;
  ac_sys_res_usage_st.ru_nswap_st = 0;
  ac_sys_res_usage_st.ru_inblock_st = 0;
  ac_sys_res_usage_st.ru_oublock_st = 0;
  ac_sys_res_usage_st.ru_msgsnd_st = 0;
  ac_sys_res_usage_st.ru_msgrcv_st = 0;
777 ac_sys_res_usage_st.ru_nsignals_st = 0;
  ac_sys_res_usage_st.ru_nvcsw_st = 0;
  ac_sys_res_usage_st.ru_nivcsw_st = 0;

  ac_cpu_percentage_st = 0;
  ac_sys_call_in_cont_st = 0;

  for (i=0; i<NO_SYS_CALLS_PLUSONE; ac_sys_call_no_st[i++] = 0);

  while (registro_apontador->prox != NULL){

    /* Calculate the difference between the current node and the next node */
    if (timeval_subtract (&diff_sys_res_usage_st.ru_utime, registro_apontador->prox->sys_res_usage.ru_utime, registro_
      perror ("variance_calc - timeval_subtract error");
      return -1;
    }
    if (timeval_subtract (&diff_sys_res_usage_st.ru_stime, registro_apontador->prox->sys_res_usage.ru_stime, registro_
      perror ("variance_calc - timeval_subtract error");
      return -1;
    }
    diff_sys_res_usage_st.ru_maxrss = registro_apontador->prox->sys_res_usage.ru_maxrss - registro_apontador->sys_res_
    diff_sys_res_usage_st.ru_ixrss = registro_apontador->prox->sys_res_usage.ru_ixrss - registro_apontador->sys_res_us
    diff_sys_res_usage_st.ru_idrss = registro_apontador->prox->sys_res_usage.ru_idrss - registro_apontador->sys_res_us
    diff_sys_res_usage_st.ru_isrss = registro_apontador->prox->sys_res_usage.ru_isrss - registro_apontador->sys_res_us
    diff_sys_res_usage_st.ru_minflt = registro_apontador->prox->sys_res_usage.ru_minflt - registro_apontador->sys_res_
    diff_sys_res_usage_st.ru_majflt = registro_apontador->prox->sys_res_usage.ru_majflt - registro_apontador->sys_res_
    diff_sys_res_usage_st.ru_nswap = registro_apontador->prox->sys_res_usage.ru_nswap - registro_apontador->sys_res_us
```

```
diff_sys_res_usage_st.ru_inblock = registro_apontador->prox->sys_res_usage.ru_inblock - registro_apontador->sys_re
diff_sys_res_usage_st.ru_oublock = registro_apontador->prox->sys_res_usage.ru_oublock - registro_apontador->sys_re
diff_sys_res_usage_st.ru_msgsnd = registro_apontador->prox->sys_res_usage.ru_msgsnd - registro_apontador->sys_res_
diff_sys_res_usage_st.ru_msgrcv = registro_apontador->prox->sys_res_usage.ru_msgrcv - registro_apontador->sys_res_
/*
  Subtract panalyser signals.
*/
(((diff_sys_res_usage_st.ru_nsignals = registro_apontador->prox->sys_res_usage.ru_nsignals - registro_apontador->s
diff_sys_res_usage_st.ru_nvcsw = registro_apontador->prox->sys_res_usage.ru_nvcsw - registro_apontador->sys_res_us
diff_sys_res_usage_st.ru_nivcsw = registro_apontador->prox->sys_res_usage.ru_nivcsw - registro_apontador->sys_res_
814
diff_sys_call_in_cont_st = registro_apontador->prox->sys_call_in_cont - registro_apontador->sys_call_in_cont;

for (i=0;i<NO_SYS_CALLS_PLUSONE;i++){
  diff_sys_call_no_st[i] = registro_apontador->prox->sys_call_no[i] - registro_apontador->sys_call_no[i];
}

/* Calculate the difference between the last difference and the mean. */
if (timeval_subtract (&diffmedia_sys_res_usage_st.ru_utime_st, diff_sys_res_usage_st.ru_utime, media_sys_res_usage
  perror ("variance_calc - timeval_subtract error");
  return -1;
}
if (timeval_subtract (&diffmedia_sys_res_usage_st.ru_stime_st, diff_sys_res_usage_st.ru_stime, media_sys_res_usage
  perror ("variance_calc - timeval_subtract error");
  return -1;
}
diffmedia_sys_res_usage_st.ru_maxrss_st = diff_sys_res_usage_st.ru_maxrss - media_sys_res_usage_st.ru_maxrss_st;
diffmedia_sys_res_usage_st.ru_ixrss_st = diff_sys_res_usage_st.ru_ixrss - media_sys_res_usage_st.ru_ixrss_st;
diffmedia_sys_res_usage_st.ru_idrss_st = diff_sys_res_usage_st.ru_idrss - media_sys_res_usage_st.ru_idrss_st;
diffmedia_sys_res_usage_st.ru_isrss_st = diff_sys_res_usage_st.ru_isrss - media_sys_res_usage_st.ru_isrss_st;
diffmedia_sys_res_usage_st.ru_minflt_st = diff_sys_res_usage_st.ru_minflt - media_sys_res_usage_st.ru_minflt_st;
diffmedia_sys_res_usage_st.ru_majflt_st = diff_sys_res_usage_st.ru_majflt - media_sys_res_usage_st.ru_majflt_st;
diffmedia_sys_res_usage_st.ru_nswap_st = diff_sys_res_usage_st.ru_nswap - media_sys_res_usage_st.ru_nswap_st;
diffmedia_sys_res_usage_st.ru_inblock_st = diff_sys_res_usage_st.ru_inblock - media_sys_res_usage_st.ru_inblock_s
diffmedia_sys_res_usage_st.ru_oublock_st = diff_sys_res_usage_st.ru_oublock - media_sys_res_usage_st.ru_oublock_s
diffmedia_sys_res_usage_st.ru_msgsnd_st = diff_sys_res_usage_st.ru_msgsnd - media_sys_res_usage_st.ru_msgsnd_st;
diffmedia_sys_res_usage_st.ru_msgrcv_st = diff_sys_res_usage_st.ru_msgrcv - media_sys_res_usage_st.ru_msgrcv_st;
diffmedia_sys_res_usage_st.ru_nsignals_st = diff_sys_res_usage_st.ru_nsignals - media_sys_res_usage_st.ru_nsignal
diffmedia_sys_res_usage_st.ru_nvcsw_st = diff_sys_res_usage_st.ru_nvcsw - media_sys_res_usage_st.ru_nvcsw_st;
diffmedia_sys_res_usage_st.ru_nivcsw_st = diff_sys_res_usage_st.ru_nivcsw - media_sys_res_usage_st.ru_nivcsw_st;

diffmedia_cpu_percentage_st = registro_apontador->prox->proc_stat.cpu_percentage - media_cpu_percentage_st;

diffmedia_sys_call_in_cont_st = diff_sys_call_in_cont_st - media_sys_call_in_cont_st;

for (i=0;i<NO_SYS_CALLS_PLUSONE;i++){
  diffmedia_sys_call_no_st[i] = diff_sys_call_no_st[i] - media_sys_call_no_st[i];
851 }

/* Calculate the square power of the last difference. */
if (timeval_pow (&pow_sys_res_usage_st.ru_utime_st, diffmedia_sys_res_usage_st.ru_utime_st, 2, debug) != 0){
  perror ("variance_calc - timeval_subtract error");
  return -1;
}
if (timeval_pow (&pow_sys_res_usage_st.ru_stime_st, diffmedia_sys_res_usage_st.ru_stime_st, 2, debug) != 0){
  perror ("variance_calc - timeval_subtract error");
  return -1;
}
pow_sys_res_usage_st.ru_maxrss_st = pow (diffmedia_sys_res_usage_st.ru_maxrss_st, 2);
pow_sys_res_usage_st.ru_ixrss_st = pow (diffmedia_sys_res_usage_st.ru_ixrss_st, 2);
pow_sys_res_usage_st.ru_idrss_st = pow (diffmedia_sys_res_usage_st.ru_idrss_st, 2);
pow_sys_res_usage_st.ru_isrss_st = pow (diffmedia_sys_res_usage_st.ru_isrss_st, 2);
pow_sys_res_usage_st.ru_minflt_st = pow (diffmedia_sys_res_usage_st.ru_minflt_st, 2);
pow_sys_res_usage_st.ru_majflt_st = pow (diffmedia_sys_res_usage_st.ru_majflt_st, 2);
pow_sys_res_usage_st.ru_nswap_st = pow (diffmedia_sys_res_usage_st.ru_nswap_st, 2);
pow_sys_res_usage_st.ru_inblock_st = pow (diffmedia_sys_res_usage_st.ru_inblock_st, 2);
pow_sys_res_usage_st.ru_oublock_st = pow (diffmedia_sys_res_usage_st.ru_oublock_st, 2);
pow_sys_res_usage_st.ru_msgsnd_st = pow (diffmedia_sys_res_usage_st.ru_msgsnd_st, 2);
pow_sys_res_usage_st.ru_msgrcv_st = pow (diffmedia_sys_res_usage_st.ru_msgrcv_st, 2);
pow_sys_res_usage_st.ru_nsignals_st = pow (diffmedia_sys_res_usage_st.ru_nsignals_st, 2);
pow_sys_res_usage_st.ru_nvcsw_st = pow (diffmedia_sys_res_usage_st.ru_nvcsw_st, 2);
pow_sys_res_usage_st.ru_nivcsw_st = pow (diffmedia_sys_res_usage_st.ru_nivcsw_st, 2);
```

```
  pow_cpu_percentage_st = pow (diffmedia_cpu_percentage_st, 2);

  pow_sys_call_in_cont_st = pow (diffmedia_sys_call_in_cont_st, 2);

  for (i=0;i<NO_SYS_CALLS_PLUSONE;i++){
    pow_sys_call_no_st[i] = pow (diffmedia_sys_call_no_st[i], 2);
  }

  /* Acumulate the difference for variance computation. */
  if (timeval_add (&ac_sys_res_usage_st.ru_utime_st, ac_sys_res_usage_st.ru_utime_st, pow_sys_res_usage_st.ru_utime
    perror ("variance_calc - timeval_add error");
888   return -1;
  }
  if (timeval_add (&ac_sys_res_usage_st.ru_stime_st, ac_sys_res_usage_st.ru_stime_st, pow_sys_res_usage_st.ru_stime
    perror ("variance_calc - timeval_add error");
    return -1;
  }
  ac_sys_res_usage_st.ru_maxrss_st = ac_sys_res_usage_st.ru_maxrss_st + pow_sys_res_usage_st.ru_maxrss_st;
  ac_sys_res_usage_st.ru_ixrss_st = ac_sys_res_usage_st.ru_ixrss_st + pow_sys_res_usage_st.ru_ixrss_st;
  ac_sys_res_usage_st.ru_idrss_st = ac_sys_res_usage_st.ru_idrss_st + pow_sys_res_usage_st.ru_idrss_st;
  ac_sys_res_usage_st.ru_isrss_st = ac_sys_res_usage_st.ru_isrss_st + pow_sys_res_usage_st.ru_isrss_st;
  ac_sys_res_usage_st.ru_minflt_st = ac_sys_res_usage_st.ru_minflt_st + pow_sys_res_usage_st.ru_minflt_st;
  ac_sys_res_usage_st.ru_majflt_st = ac_sys_res_usage_st.ru_majflt_st + pow_sys_res_usage_st.ru_majflt_st;
  ac_sys_res_usage_st.ru_nswap_st = ac_sys_res_usage_st.ru_nswap_st + pow_sys_res_usage_st.ru_nswap_st;
  ac_sys_res_usage_st.ru_inblock_st = ac_sys_res_usage_st.ru_inblock_st + pow_sys_res_usage_st.ru_inblock_st;
  ac_sys_res_usage_st.ru_oublock_st = ac_sys_res_usage_st.ru_oublock_st +  pow_sys_res_usage_st.ru_oublock_st;
  ac_sys_res_usage_st.ru_msgsnd_st = ac_sys_res_usage_st.ru_msgsnd_st + pow_sys_res_usage_st.ru_msgsnd_st;
  ac_sys_res_usage_st.ru_msgrcv_st = ac_sys_res_usage_st.ru_msgrcv_st + pow_sys_res_usage_st.ru_msgrcv_st;
  ac_sys_res_usage_st.ru_nsignals_st = ac_sys_res_usage_st.ru_nsignals_st + pow_sys_res_usage_st.ru_nsignals_st;
  ac_sys_res_usage_st.ru_nvcsw_st = ac_sys_res_usage_st.ru_nvcsw_st + pow_sys_res_usage_st.ru_nvcsw_st;
  ac_sys_res_usage_st.ru_nivcsw_st = ac_sys_res_usage_st.ru_nivcsw_st + pow_sys_res_usage_st.ru_nivcsw_st;

  ac_cpu_percentage_st = ac_cpu_percentage_st + pow_cpu_percentage_st;

  ac_sys_call_in_cont_st = ac_sys_call_in_cont_st + pow_sys_call_in_cont_st;

  for (i=0;i<NO_SYS_CALLS_PLUSONE;i++){
    ac_sys_call_no_st[i] = ac_sys_call_no_st[i] + pow_sys_call_no_st[i];
  }

  registro_apontador = registro_apontador->prox;
}

/* Calculate variance */
divisor = registro_apontador->indice;
if (timeval_div (&sys_res_usage_st->ru_utime_st, ac_sys_res_usage_st.ru_utime_st, divisor, debug) != 0){
  perror ("variance_calc - timeval_div error");
  return -1;
925 }
if (timeval_div (&sys_res_usage_st->ru_stime_st, ac_sys_res_usage_st.ru_stime_st, divisor, debug) != 0){
  perror ("variance_calc - timeval_div error");
  return -1;
}
sys_res_usage_st->ru_maxrss_st = ac_sys_res_usage_st.ru_maxrss_st / divisor;
sys_res_usage_st->ru_ixrss_st = ac_sys_res_usage_st.ru_ixrss_st / divisor;
sys_res_usage_st->ru_idrss_st = ac_sys_res_usage_st.ru_idrss_st / divisor;
sys_res_usage_st->ru_isrss_st = ac_sys_res_usage_st.ru_isrss_st / divisor;
sys_res_usage_st->ru_minflt_st = ac_sys_res_usage_st.ru_minflt_st / divisor;
sys_res_usage_st->ru_majflt_st = ac_sys_res_usage_st.ru_majflt_st / divisor;
sys_res_usage_st->ru_nswap_st = ac_sys_res_usage_st.ru_nswap_st / divisor;
sys_res_usage_st->ru_inblock_st = ac_sys_res_usage_st.ru_inblock_st / divisor;
sys_res_usage_st->ru_oublock_st = ac_sys_res_usage_st.ru_oublock_st / divisor;
sys_res_usage_st->ru_msgsnd_st = ac_sys_res_usage_st.ru_msgsnd_st / divisor;
sys_res_usage_st->ru_msgrcv_st = ac_sys_res_usage_st.ru_msgrcv_st / divisor;
sys_res_usage_st->ru_nsignals_st = ac_sys_res_usage_st.ru_nsignals_st / divisor;
sys_res_usage_st->ru_nvcsw_st = ac_sys_res_usage_st.ru_nvcsw_st / divisor;
sys_res_usage_st->ru_nivcsw_st = ac_sys_res_usage_st.ru_nivcsw_st / divisor;

*cpu_percentage_st = ac_cpu_percentage_st / divisor;

*sys_call_in_cont_st = ac_sys_call_in_cont_st / divisor;

for (i=0;i<NO_SYS_CALLS_PLUSONE;i++){
```

```
    sys_call_no_st[i] = ac_sys_call_no_st[i] / divisor;
  }

  return 0;
}


int order_list (lista_db lista, lista_db *lista_ordenada, unsigned int n_registros, unsigned short int criterio_ord, i
     /*
        Buble sort algorithm for ordenation. criterio_ord defines the order
        criteria.
     */
962 {

  lista_db registro_apontador;
  int j, pass;
  unsigned int switched = TRUE;

  unsigned int i;
  unsigned long int hold_indice;
  double hold_momento_insercao;
  struct rusage hold_sys_res_usage;
  unsigned long int hold_sys_call_in_cont;
  unsigned long int hold_sys_call_no[NO_SYS_CALLS_PLUSONE];
  struct timeval hold_diff_ru_utime;
  struct timeval hold_diff_ru_stime;
  unsigned long int hold_diff_sys_call_in_cont;

  if (debug)
    fprintf (stderr, "\norder_list - criterio_ord:%u\n", criterio_ord);

  for (pass = 0;((pass < (n_registros)) && (switched));pass++) {
    /* external loop controls the passage */
    switched = FALSE;  /* no sort in this passage */

    registro_apontador = lista;
    for (j = 0; j < (n_registros - pass); j++){
      /* internal loop control the individual passage */
      if (criterio_ord == NSYSCALLS){
        if (registro_apontador->diff_sys_call_in_cont > registro_apontador->prox->diff_sys_call_in_cont){
          /* elements out of order, sort necessary */
          switched = TRUE;
          hold_indice = registro_apontador->indice;
          hold_momento_insercao = registro_apontador->momento_insercao;
          hold_sys_res_usage = registro_apontador->sys_res_usage;
          hold_sys_call_in_cont = registro_apontador->sys_call_in_cont;
          for (i=0;i<NO_SYS_CALLS_PLUSONE;i++){
            hold_sys_call_no[i] = registro_apontador->sys_call_no[i];
          }
999       hold_diff_ru_utime = registro_apontador->diff_ru_utime;
          hold_diff_ru_stime = registro_apontador->diff_ru_stime;
          hold_diff_sys_call_in_cont = registro_apontador->diff_sys_call_in_cont;

          registro_apontador->indice = registro_apontador->prox->indice;
          registro_apontador->momento_insercao = registro_apontador->prox->momento_insercao;
          registro_apontador->sys_res_usage = registro_apontador->prox->sys_res_usage;
          registro_apontador->sys_call_in_cont = registro_apontador->prox->sys_call_in_cont;
          for (i=0;i<NO_SYS_CALLS_PLUSONE;i++){
            registro_apontador->sys_call_no[i] = registro_apontador->prox->sys_call_no[i];
          }
          registro_apontador->diff_ru_utime = registro_apontador->prox->diff_ru_utime;
          registro_apontador->diff_ru_stime = registro_apontador->prox->diff_ru_stime;
          registro_apontador->diff_sys_call_in_cont = registro_apontador->prox->diff_sys_call_in_cont;

          registro_apontador->prox->indice = hold_indice;
          registro_apontador->prox->momento_insercao = hold_momento_insercao;
          registro_apontador->prox->sys_res_usage = hold_sys_res_usage;
          registro_apontador->prox->sys_call_in_cont = hold_sys_call_in_cont;
          for (i=0;i<NO_SYS_CALLS_PLUSONE;i++){
            registro_apontador->prox->sys_call_no[i] = hold_sys_call_no[i];
          }
          registro_apontador->prox->diff_ru_utime = hold_diff_ru_utime;
          registro_apontador->prox->diff_ru_stime = hold_diff_ru_stime;
```

```
      registro_apontador->prox->diff_sys_call_in_cont = hold_diff_sys_call_in_cont;

    } /* end of if */
  } /* if (criterio_ord == NSYSCALLS) */
  else if (criterio_ord == UTIME){
    if ((registro_apontador->diff_ru_utime.tv_sec * M + registro_apontador->diff_ru_utime.tv_usec) > (registro_apo
      /* elements out of order, sort necessary */
      switched = TRUE;
      hold_indice = registro_apontador->indice;
      hold_momento_insercao = registro_apontador->momento_insercao;
      hold_sys_res_usage = registro_apontador->sys_res_usage;
      hold_sys_call_in_cont = registro_apontador->sys_call_in_cont;
      for (i=0;i<NO_SYS_CALLS_PLUSONE;i++){
1036    hold_sys_call_no[i] = registro_apontador->sys_call_no[i];
      }
      hold_diff_ru_utime = registro_apontador->diff_ru_utime;
      hold_diff_ru_stime = registro_apontador->diff_ru_stime;
      hold_diff_sys_call_in_cont = registro_apontador->diff_sys_call_in_cont;

      registro_apontador->indice = registro_apontador->prox->indice;
      registro_apontador->momento_insercao = registro_apontador->prox->momento_insercao;
      registro_apontador->sys_res_usage = registro_apontador->prox->sys_res_usage;
      registro_apontador->sys_call_in_cont = registro_apontador->prox->sys_call_in_cont;
      for (i=0;i<NO_SYS_CALLS_PLUSONE;i++){
        registro_apontador->sys_call_no[i] = registro_apontador->prox->sys_call_no[i];
      }
      registro_apontador->diff_ru_utime = registro_apontador->prox->diff_ru_utime;
      registro_apontador->diff_ru_stime = registro_apontador->prox->diff_ru_stime;
      registro_apontador->diff_sys_call_in_cont = registro_apontador->prox->diff_sys_call_in_cont;

      registro_apontador->prox->indice = hold_indice;
      registro_apontador->prox->momento_insercao = hold_momento_insercao;
      registro_apontador->prox->sys_res_usage = hold_sys_res_usage;
      registro_apontador->prox->sys_call_in_cont = hold_sys_call_in_cont;
      for (i=0;i<NO_SYS_CALLS_PLUSONE;i++){
        registro_apontador->prox->sys_call_no[i] = hold_sys_call_no[i];
      }
      registro_apontador->prox->diff_ru_utime = hold_diff_ru_utime;
      registro_apontador->prox->diff_ru_stime = hold_diff_ru_stime;
      registro_apontador->prox->diff_sys_call_in_cont = hold_diff_sys_call_in_cont;

    } /* end of if */
  }/* if (criterio_ord == UTIME) */
  else{
    if ((registro_apontador->diff_ru_stime.tv_sec * M + registro_apontador->diff_ru_stime.tv_usec) > (registro_ap
      /* elements out of order, sort necessary */
      switched = TRUE;
      hold_indice = registro_apontador->indice;
      hold_momento_insercao = registro_apontador->momento_insercao;
      hold_sys_res_usage = registro_apontador->sys_res_usage;
1073  hold_sys_call_in_cont = registro_apontador->sys_call_in_cont;
      for (i=0;i<NO_SYS_CALLS_PLUSONE;i++){
        hold_sys_call_no[i] = registro_apontador->sys_call_no[i];
      }
      hold_diff_ru_utime = registro_apontador->diff_ru_utime;
      hold_diff_ru_stime = registro_apontador->diff_ru_stime;
      hold_diff_sys_call_in_cont = registro_apontador->diff_sys_call_in_cont;

      registro_apontador->indice = registro_apontador->prox->indice;
      registro_apontador->momento_insercao = registro_apontador->prox->momento_insercao;
      registro_apontador->sys_res_usage = registro_apontador->prox->sys_res_usage;
      registro_apontador->sys_call_in_cont = registro_apontador->prox->sys_call_in_cont;
      for (i=0;i<NO_SYS_CALLS_PLUSONE;i++){
        registro_apontador->sys_call_no[i] = registro_apontador->prox->sys_call_no[i];
      }
      registro_apontador->diff_ru_utime = registro_apontador->prox->diff_ru_utime;
      registro_apontador->diff_ru_stime = registro_apontador->prox->diff_ru_stime;
      registro_apontador->diff_sys_call_in_cont = registro_apontador->prox->diff_sys_call_in_cont;

      registro_apontador->prox->indice = hold_indice;
      registro_apontador->prox->momento_insercao = hold_momento_insercao;
      registro_apontador->prox->sys_res_usage = hold_sys_res_usage;
      registro_apontador->prox->sys_call_in_cont = hold_sys_call_in_cont;
```

```
            for (i=0;i<NO_SYS_CALLS_PLUSONE;i++){
              registro_apontador->prox->sys_call_no[i] = hold_sys_call_no[i];
            }
            registro_apontador->prox->diff_ru_utime = hold_diff_ru_utime;
            registro_apontador->prox->diff_ru_stime = hold_diff_ru_stime;
            registro_apontador->prox->diff_sys_call_in_cont = hold_diff_sys_call_in_cont;

          } /* end of if */
        } /* end of else */
        registro_apontador = registro_apontador->prox;
      } /* end of internal loop */
    } /* end o external loop */

    * lista_ordenada = lista;
1110 return 0;
}


double median_calc (lista_db lista, unsigned short int criterio_ord, int debug)

{
  unsigned int i;
  lista_db lista_ordenada, registro_apontador, registro_fim;
  div_t divisao;
  double valor1 = 0, valor2 = 0, mediana = 0;
  unsigned long int n_registros;

  init_list (&lista_ordenada);

  registro_apontador = lista;
  registro_fim = localizar_fim (registro_apontador);
  n_registros = registro_fim->indice;

  if (debug){
    fprintf (stderr, "\nmedian_calc - n_registros:%lu\n", n_registros);
    fprintf (stderr, "median_calc - criterio_ord:%u\n", criterio_ord);
  }

  order_list (lista, &lista_ordenada, n_registros, criterio_ord, debug);

  divisao = div (n_registros,2);

  if (debug){
    fprintf (stderr, "\nmedian_calc - divisao.rem:%d\n", divisao.rem);
    fprintf (stderr, "median_calc - divisao.quot:%d\n", divisao.quot);
  }

  if (divisao.rem > 0){
    for (i=0; i <= divisao.quot; i++){
      if (lista_ordenada != NULL){
        lista_ordenada = lista_ordenada->prox;
1147    }
      else {
        perror ("median_calc - number of registers > values list error");
        return -1;
      }

      if (debug){
        fprintf (stderr, "\nmedian_calc - lista_ordenada user time:%.15f\n", (((double) lista_ordenada->diff_ru_utime
        fprintf (stderr, "median_calc - lista_ordenada system time:%.15f\n", (((double) lista_ordenada->diff_ru_stime
        fprintf (stderr, "median_calc - lista_ordenada system calls:%.15f\n", ((double) lista_ordenada->diff_sys_call
      }
    }
    if (criterio_ord == NSYSCALLS){
      mediana = (double) lista_ordenada->diff_sys_call_in_cont;
    } /* if (criterio_ord == NSYSCALLS) */
    else if (criterio_ord == UTIME){
      mediana = (((double) lista_ordenada->diff_ru_utime.tv_sec) * M + lista_ordenada->diff_ru_utime.tv_usec);
    }/* if (criterio_ord == UTIME) */
    else{
      mediana = (((double) lista_ordenada->diff_ru_stime.tv_sec) * M + lista_ordenada->diff_ru_stime.tv_usec);
    }
  }
```

```
    else{
      for (i=0; i < divisao.quot; i++){
        if (lista_ordenada != NULL){
          lista_ordenada = lista_ordenada->prox;
        }
        else {
          perror ("median_calc - number of registers > values list error");
          return -1;
        }

        if (debug){
          fprintf (stderr, "\nmedian_calc - lista_ordenada user time:%.15f\n", (((double) lista_ordenada->diff_ru_utime.
          fprintf (stderr, "median_calc - lista_ordenada system time:%.15f\n", (((double) lista_ordenada->diff_ru_stime.
          fprintf (stderr, "median_calc - lista_ordenada system calls:%.15f\n", ((double) lista_ordenada->diff_sys_call_
        }
1184
      }
      if (criterio_ord == NSYSCALLS){
        valor1 = (double) lista_ordenada->diff_sys_call_in_cont;
        valor2 = (double) lista_ordenada->prox->diff_sys_call_in_cont;
        mediana = (valor1 + valor2) / 2;

        if (debug){
          fprintf (stderr, "\nmedian_calc - diff_sys_call_in_cont valor1:%.15f\n", valor1);
          fprintf (stderr, "median_calc - diff_sys_call_in_cont valor2:%.15f\n", valor2);
        }

      } /* if (criterio_ord == NSYSCALLS) */
      else if (criterio_ord == UTIME){
        valor1 = (((double) lista_ordenada->diff_ru_utime.tv_sec) * M + lista_ordenada->diff_ru_utime.tv_usec);
        valor2 = (((double) lista_ordenada->prox->diff_ru_utime.tv_sec) * M + lista_ordenada->prox->diff_ru_utime.tv_us
        mediana = (valor1 + valor2) / 2;

        if (debug){
          fprintf (stderr, "\nmedian_calc - diff_ru_utime valor1:%.15f\n", valor1);
          fprintf (stderr, "median_calc - diff_ru_utime valor2:%.15f\n", valor2);
        }

      }/* if (criterio_ord == UTIME) */
      else{
        valor1 = (((double) lista_ordenada->diff_ru_stime.tv_sec) * M + lista_ordenada->diff_ru_stime.tv_usec);
        valor2 = (((double) lista_ordenada->prox->diff_ru_stime.tv_sec) * M + lista_ordenada->prox->diff_ru_stime.tv_us
        mediana = (valor1 + valor2) / 2;

        if (debug){
        fprintf (stderr, "\nmedian_calc - diff_ru_stime valor1:%.15f\n", valor1);
        fprintf (stderr, "median_calc - diff_ru_stime valor2:%.15f\n", valor2);
        }

      }
    }
    return (mediana);
1221 }


int calc_present (lista_db lista, unsigned short int flag, unsigned short int criterio_ord, FILE *file, FILE *stfile,
{
  double mediana;
  struct statistic_rusage media_sys_res_usage_st, variancia_sys_res_usage_st, desviopadrao_sys_res_usage_st;
  double media_cpu_percentage_st, variancia_cpu_percentage_st, desviopadrao_cpu_percentage_st;
  double media_sys_call_in_cont_st, variancia_sys_call_in_cont_st, desviopadrao_sys_call_in_cont_st;
  double media_sys_call_no_st[NO_SYS_CALLS_PLUSONE], variancia_sys_call_no_st[NO_SYS_CALLS_PLUSONE], desviopadrao_sy
  char *sys_call_name[NO_SYS_CALLS_PLUSONE];
  lista_db lista_ordenada, registro_apontador, registro_apontador_ord, registro_fim, registro_fim_ord;
  unsigned int i, n_registros;
  double twu_media, twu_desviopadrao, twu_coefvariacao, tws_media, tws_desviopadrao, tws_coefvariacao;

  media_sys_res_usage_st.ru_utime_st.tv_usec = 0;
  media_sys_res_usage_st.ru_utime_st.tv_sec = 0;
  media_sys_res_usage_st.ru_stime_st.tv_usec = 0;
  media_sys_res_usage_st.ru_stime_st.tv_sec = 0;
  media_sys_res_usage_st.ru_maxrss_st = 0;
  media_sys_res_usage_st.ru_ixrss_st = 0;
```

```
media_sys_res_usage_st.ru_idrss_st = 0;
media_sys_res_usage_st.ru_isrss_st = 0;
media_sys_res_usage_st.ru_minflt_st = 0;
media_sys_res_usage_st.ru_majflt_st = 0;
media_sys_res_usage_st.ru_nswap_st = 0;
media_sys_res_usage_st.ru_inblock_st = 0;
media_sys_res_usage_st.ru_oublock_st = 0;
media_sys_res_usage_st.ru_msgsnd_st = 0;
media_sys_res_usage_st.ru_msgrcv_st = 0;
media_sys_res_usage_st.ru_nsignals_st = 0;
media_sys_res_usage_st.ru_nvcsw_st = 0;
media_sys_res_usage_st.ru_nivcsw_st = 0;

desviopadrao_sys_res_usage_st.ru_utime_st.tv_usec = 0;
desviopadrao_sys_res_usage_st.ru_utime_st.tv_sec = 0;
desviopadrao_sys_res_usage_st.ru_stime_st.tv_usec = 0;
desviopadrao_sys_res_usage_st.ru_stime_st.tv_sec = 0;
desviopadrao_sys_res_usage_st.ru_maxrss_st = 0;
desviopadrao_sys_res_usage_st.ru_ixrss_st = 0;
desviopadrao_sys_res_usage_st.ru_idrss_st = 0;
desviopadrao_sys_res_usage_st.ru_isrss_st = 0;
desviopadrao_sys_res_usage_st.ru_minflt_st = 0;
desviopadrao_sys_res_usage_st.ru_majflt_st = 0;
desviopadrao_sys_res_usage_st.ru_nswap_st = 0;
desviopadrao_sys_res_usage_st.ru_inblock_st = 0;
desviopadrao_sys_res_usage_st.ru_oublock_st = 0;
desviopadrao_sys_res_usage_st.ru_msgsnd_st = 0;
desviopadrao_sys_res_usage_st.ru_msgrcv_st = 0;
desviopadrao_sys_res_usage_st.ru_nsignals_st = 0;
desviopadrao_sys_res_usage_st.ru_nvcsw_st = 0;
desviopadrao_sys_res_usage_st.ru_nivcsw_st = 0;

media_cpu_percentage_st = 0;
desviopadrao_cpu_percentage_st = 0;

media_sys_call_in_cont_st = 0;
desviopadrao_sys_call_in_cont_st = 0;

for (i=0; i<NO_SYS_CALLS_PLUSONE; i++){
  media_sys_call_no_st[i] = 0;
  desviopadrao_sys_call_no_st[i] = 0;
}

sys_call_name_init (sys_call_name);

if (flag & STATISTICS){
  if (mean_calc (lista, &media_sys_res_usage_st, &media_cpu_percentage_st, &media_sys_call_in_cont_st, media_sys_ca
    perror ("calc_present - mean_calc error");
    return -1;
  }

  if (variance_calc (lista, &variancia_sys_res_usage_st, &variancia_cpu_percentage_st, &variancia_sys_call_in_cont_
    perror ("calc_present - variance_calc error");
    return -1;
  }

  /* Calculate standard deviation operating the square root over variance. */
  if (timeval_sqrt (&desviopadrao_sys_res_usage_st.ru_utime_st, variancia_sys_res_usage_st.ru_utime_st, debug) != (
    perror ("calc_present - timeval_subtract error");
    return -1;
  }
  if (timeval_sqrt (&desviopadrao_sys_res_usage_st.ru_stime_st, variancia_sys_res_usage_st.ru_stime_st, debug) != (
    perror ("calc_present - timeval_subtract error");
    return -1;
  }
  desviopadrao_sys_res_usage_st.ru_maxrss_st = sqrt (variancia_sys_res_usage_st.ru_maxrss_st);
  desviopadrao_sys_res_usage_st.ru_ixrss_st = sqrt (variancia_sys_res_usage_st.ru_ixrss_st);
  desviopadrao_sys_res_usage_st.ru_idrss_st = sqrt (variancia_sys_res_usage_st.ru_idrss_st);
  desviopadrao_sys_res_usage_st.ru_isrss_st = sqrt (variancia_sys_res_usage_st.ru_isrss_st);
  desviopadrao_sys_res_usage_st.ru_minflt_st = sqrt (variancia_sys_res_usage_st.ru_minflt_st);
  desviopadrao_sys_res_usage_st.ru_majflt_st = sqrt (variancia_sys_res_usage_st.ru_majflt_st);
  desviopadrao_sys_res_usage_st.ru_nswap_st = sqrt (variancia_sys_res_usage_st.ru_nswap_st);
  desviopadrao_sys_res_usage_st.ru_inblock_st = sqrt (variancia_sys_res_usage_st.ru_inblock_st);
```

```
  desviopadrao_sys_res_usage_st.ru_oublock_st = sqrt (variancia_sys_res_usage_st.ru_oublock_st);
  desviopadrao_sys_res_usage_st.ru_msgsnd_st = sqrt (variancia_sys_res_usage_st.ru_msgsnd_st);
  desviopadrao_sys_res_usage_st.ru_msgrcv_st = sqrt (variancia_sys_res_usage_st.ru_msgrcv_st);
  desviopadrao_sys_res_usage_st.ru_nsignals_st = sqrt (variancia_sys_res_usage_st.ru_nsignals_st);
  desviopadrao_sys_res_usage_st.ru_nvcsw_st = sqrt (variancia_sys_res_usage_st.ru_nvcsw_st);
  desviopadrao_sys_res_usage_st.ru_nivcsw_st = sqrt (variancia_sys_res_usage_st.ru_nivcsw_st);

  desviopadrao_cpu_percentage_st = sqrt (variancia_cpu_percentage_st);

  desviopadrao_sys_call_in_cont_st = sqrt (variancia_sys_call_in_cont_st);

  for (i=0;i<NO_SYS_CALLS_PLUSONE;i++){
    desviopadrao_sys_call_no_st[i] = sqrt (variancia_sys_call_no_st[i]);
  }
} /* if (flag & STATISTICS) */

if (print_list (lista, flag, sys_call_name, media_sys_res_usage_st, desviopadrao_sys_res_usage_st, media_cpu_percent
1332  perror ("calc_present - print_list error");
  return -1;
}

if (flag & STATISTICS){

  if (debug)
    fprintf (stderr, "\ncalc_present - criterio_ord:%u\n", criterio_ord);
  mediana = median_calc(lista, criterio_ord, debug);

  init_list (&lista_ordenada);

  registro_apontador = lista;
  registro_fim = localizar_fim (registro_apontador);
  n_registros = registro_fim->indice;

  if (debug){
    fprintf (stderr, "\ncalc_present - lista->indice:%lu\n", registro_apontador->indice);
  }

  order_list (lista, &lista_ordenada, n_registros, criterio_ord, debug);
  registro_apontador_ord = lista_ordenada;
  registro_fim_ord = localizar_fim (registro_apontador_ord);

  if (debug){
    fprintf (stderr, "\ncalc_present - lista_ordenada->indice:%lu\n", registro_apontador_ord->indice);
  }

  if (criterio_ord == NSYSCALLS){
    fprintf (file, "median of system calls difference values:%.15f\n", mediana);
    fprintf (file, "values range: %ld a %ld\n", lista_ordenada->diff_sys_call_in_cont, registro_fim_ord->diff_sys_c
  } /* if (criterio_ord == NSYSCALLS) */
  else if (criterio_ord == UTIME){
    fprintf (file, "median of user time used difference values:%.15f\n", mediana);
    fprintf (file, "values range: %ld a %ld\n", (lista_ordenada->diff_ru_utime.tv_sec * M + lista_ordenada->diff_ru
  }/* if (criterio_ord == UTIME) */
  else{
1369  fprintf (file, "median of system time used difference values:%.15f\n", mediana);
    fprintf (file, "values range: %ld a %ld\n", (lista_ordenada->diff_ru_stime.tv_sec * M + lista_ordenada->diff_ru
  }

  twu_media = (((double) media_sys_res_usage_st.ru_utime_st.tv_sec) * M + media_sys_res_usage_st.ru_utime_st.tv_us
  twu_desviopadrao = (((double) desviopadrao_sys_res_usage_st.ru_utime_st.tv_sec) * M + desviopadrao_sys_res_usage_
  ((twu_media > 0) ? (twu_coefvariacao = twu_desviopadrao / twu_media) : (twu_coefvariacao = 0));
  tws_media = (((double) media_sys_res_usage_st.ru_stime_st.tv_sec) * M + media_sys_res_usage_st.ru_stime_st.tv_us
  tws_desviopadrao = (((double) desviopadrao_sys_res_usage_st.ru_stime_st.tv_sec) * M + desviopadrao_sys_res_usage_
  ((tws_media > 0) ? (tws_coefvariacao = tws_desviopadrao / tws_media) : (tws_coefvariacao = 0));

  ((twu_coefvariacao > 0) ? fprintf (file, "user time used coefficient of variation:%.15f\n", twu_coefvariacao) :
  ((tws_coefvariacao > 0) ? fprintf (file, "system time used coefficient of variation:%.15f\n", tws_coefvariacao)
  ((media_sys_res_usage_st.ru_maxrss_st > 0) ? fprintf (file, "maximum resident set size coefficient of variation:
  ((media_sys_res_usage_st.ru_ixrss_st > 0) ? fprintf (file, "integral shared memory size coefficient of variation
  ((media_sys_res_usage_st.ru_idrss_st > 0) ? fprintf (file, "integral unshared data size coefficient of variation
  ((media_sys_res_usage_st.ru_isrss_st > 0) ? fprintf (file, "integral unshared stack size coefficient of variatio
  ((media_sys_res_usage_st.ru_minflt_st > 0) ? fprintf (file, "page reclaims coefficient of variation:%.15f\n", de
  ((media_sys_res_usage_st.ru_majflt_st > 0) ? fprintf (file, "page faults coefficient of variation:%.15f\n", desv
```

```
    ((media_sys_res_usage_st.ru_nswap_st > 0) ? fprintf (file, "swaps coefficient of variation:%.15f\n", desviopadrao_
    ((media_sys_res_usage_st.ru_inblock_st > 0) ? fprintf (file, "block input operations coefficient of variation:%.15
    ((media_sys_res_usage_st.ru_oublock_st > 0) ? fprintf (file, "block output operations coefficient of variation:%.1
    ((media_sys_res_usage_st.ru_msgsnd_st > 0) ? fprintf (file, "messages sent coefficient of variation:%.15f\n", desv
    ((media_sys_res_usage_st.ru_msgrcv_st > 0) ? fprintf (file, "messages received coefficient of variation:%.15f\n",
    ((media_sys_res_usage_st.ru_nsignals_st > 0) ? fprintf (file, "signals received coefficient of variation:%.15f\n",
    ((media_sys_res_usage_st.ru_nvcsw_st > 0) ? fprintf (file, "voluntary context switches coefficient of variation:%.
    ((media_sys_res_usage_st.ru_nivcsw_st > 0) ? fprintf (file, "involuntary context switches coefficient of variation
    ((media_cpu_percentage_st > 0) ? fprintf (file, "CPU percentage coefficient of variation:%.15f\n", desviopadrao_cp
    ((media_sys_call_in_cont_st > 0) ? fprintf (file, "number of system calls coefficient of variation:%.15f\n", desvi

    for (i=1; i<NO_SYS_CALLS_PLUSONE; i++){
      ((media_sys_call_no_st[i] > 0) ? fprintf (file, "%s coefficient of variation:%.15f\n", sys_call_name[i], desviop
    }
  }/* if (flag & STATISTICS) */
  fflush (file);

  return 0;
}

int
print_getrusage_summary (struct rusage process_usage, struct statistic_rusage process_usage_media, struct statistic_ru
{
  double twu, tws, twu_media, tws_media, twu_desviopadrao, tws_desviopadrao, twu_diff, tws_diff;

  if (debug)
    fprintf (stderr, "\nprint_getrusage_summary - %u \n", i);

  twu = (((double) process_usage.ru_utime.tv_sec) * M + process_usage.ru_utime.tv_usec);
  twu_media = (((double) process_usage_media.ru_utime_st.tv_sec) * M + process_usage_media.ru_utime_st.tv_usec);
  twu_desviopadrao = (((double) process_usage_desviopadrao.ru_utime_st.tv_sec) * M + process_usage_desviopadrao.ru_ut
  twu_diff = (((double) process_usage_diff.ru_utime.tv_sec) * M + process_usage_diff.ru_utime.tv_usec);
  tws = (((double) process_usage.ru_stime.tv_sec) * M + process_usage.ru_stime.tv_usec);
  tws_media = (((double) process_usage_media.ru_stime_st.tv_sec) * M + process_usage_media.ru_stime_st.tv_usec);
  tws_desviopadrao = (((double) process_usage_desviopadrao.ru_stime_st.tv_sec) * M + process_usage_desviopadrao.ru_st
  tws_diff = (((double) process_usage_diff.ru_stime.tv_sec) * M + process_usage_diff.ru_stime.tv_usec);


  (((twu > 0) || (twu_media > 0) || (twu_desviopadrao > 0)) ? fprintf (file, "user time used:%.15fus\n", twu) : 0);
  (((twu > 0) || (twu_media > 0) || (twu_desviopadrao > 0)) ? fprintf (file, "user time used difference:%.15fus\n", t
  ((twu_media > 0) ? fprintf (file, "user time used mean:%.15fus\n", twu_media) : 0);
  ((twu_desviopadrao > 0) ? fprintf (file, "user time used std deviation:%.15fus\n", twu_desviopadrao) : 0);

  (((tws > 0) || (tws_media > 0) || (tws_desviopadrao > 0)) ? fprintf (file, "system time used:%.15fus\n", tws) : 0);
  (((tws > 0) || (tws_media > 0) || (tws_desviopadrao > 0)) ? fprintf (file, "system time used difference:%.15fus\n",
  ((tws_media > 0) ? fprintf (file, "system time used mean:%.15fus\n", tws_media) : 0);
  ((tws_desviopadrao > 0) ? fprintf (file, "system time used std deviation:%.15fus\n", tws_desviopadrao) : 0);

  (((process_usage.ru_maxrss > 0) || (process_usage_media.ru_maxrss_st > 0) || (process_usage_desviopadrao.ru_maxrss_
  (((process_usage.ru_maxrss > 0) || (process_usage_media.ru_maxrss_st > 0) || (process_usage_desviopadrao.ru_maxrss_
  ((process_usage_media.ru_maxrss_st > 0) ? fprintf (file, "maximum resident set size mean:%.15f\n", process_usage_me
  ((process_usage_desviopadrao.ru_maxrss_st > 0) ? fprintf (file, "maximum resident set size std deviation:%.15f\n",

  (((process_usage.ru_ixrss > 0) || (process_usage_media.ru_ixrss_st > 0) || (process_usage_desviopadrao.ru_ixrss_st
  (((process_usage.ru_ixrss > 0) || (process_usage_media.ru_ixrss_st > 0) || (process_usage_desviopadrao.ru_ixrss_st
1443 ((process_usage_media.ru_ixrss_st > 0) ? fprintf (file, "integral shared memory size mean:%.15f\n", process_usage_m
  ((process_usage_desviopadrao.ru_ixrss_st > 0) ? fprintf (file, "integral shared memory size std deviation:%.15f\n",

  (((process_usage.ru_idrss > 0) || (process_usage_media.ru_idrss_st > 0) || (process_usage_desviopadrao.ru_idrss_st
  (((process_usage.ru_idrss > 0) || (process_usage_media.ru_idrss_st > 0) || (process_usage_desviopadrao.ru_idrss_st
  ((process_usage_media.ru_idrss_st > 0) ? fprintf (file, "integral unshared data size mean:%.15f\n", process_usage_m
  ((process_usage_desviopadrao.ru_idrss_st > 0) ? fprintf (file, "integral unshared data size std deviation:%.15f\n",

  (((process_usage.ru_isrss > 0) || (process_usage_media.ru_isrss_st > 0) || (process_usage_desviopadrao.ru_isrss_st
  (((process_usage.ru_isrss > 0) || (process_usage_media.ru_isrss_st > 0) || (process_usage_desviopadrao.ru_isrss_st
  ((process_usage_media.ru_isrss_st > 0) ? fprintf (file, "integral unshared stack size mean:%.15f\n", process_usage_
  ((process_usage_desviopadrao.ru_isrss_st > 0) ? fprintf (file, "integral unshared stack size std deviation:%.15f\n'

  (((process_usage.ru_minflt > 0) || (process_usage_media.ru_minflt_st > 0) || (process_usage_desviopadrao.ru_minflt_
  (((process_usage.ru_minflt > 0) || (process_usage_media.ru_minflt_st > 0) || (process_usage_desviopadrao.ru_minflt_
  ((process_usage_media.ru_minflt_st > 0) ? fprintf (file, "page reclaims mean:%.15f\n", process_usage_media.ru_minfl
  ((process_usage_desviopadrao.ru_minflt_st > 0) ? fprintf (file, "page reclaims std deviation:%.15f\n", process_usa
```

```
  (((process_usage.ru_majflt > 0) || (process_usage_media.ru_majflt_st > 0) || (process_usage_desviopadrao.ru_majflt_s
  (((process_usage.ru_majflt > 0) || (process_usage_media.ru_majflt_st > 0) || (process_usage_desviopadrao.ru_majflt_s
  ((process_usage_media.ru_majflt_st > 0) ? fprintf (file, "page faults mean:%.15f\n", process_usage_media.ru_majflt_s
  ((process_usage_desviopadrao.ru_majflt_st > 0) ? fprintf (file, "page faults std deviation:%.15f\n", process_usage_d

  (((process_usage.ru_nswap > 0) || (process_usage_media.ru_nswap_st > 0) || (process_usage_desviopadrao.ru_nswap_st >
  (((process_usage.ru_nswap > 0) || (process_usage_media.ru_nswap_st > 0) || (process_usage_desviopadrao.ru_nswap_st >
  ((process_usage_media.ru_nswap_st > 0) ? fprintf (file, "swaps mean:%.15f\n", process_usage_media.ru_nswap_st) : 0);
  ((process_usage_desviopadrao.ru_nswap_st > 0) ? fprintf (file, "swaps std deviation:%.15f\n", process_usage_desviopa

  (((process_usage.ru_inblock > 0) || (process_usage_media.ru_inblock_st > 0) || (process_usage_desviopadrao.ru_inbloc
  (((process_usage.ru_inblock > 0) || (process_usage_media.ru_inblock_st > 0) || (process_usage_desviopadrao.ru_inbloc
  ((process_usage_media.ru_inblock_st > 0) ? fprintf (file, "block input operations mean:%.15f\n", process_usage_media
  ((process_usage_desviopadrao.ru_inblock_st > 0) ? fprintf (file, "block input operations std deviation:%.15f\n", pro

  (((process_usage.ru_oublock > 0) || (process_usage_media.ru_oublock_st > 0) || (process_usage_desviopadrao.ru_oubloc
  (((process_usage.ru_oublock > 0) || (process_usage_media.ru_oublock_st > 0) || (process_usage_desviopadrao.ru_oubloc
  ((process_usage_media.ru_oublock_st > 0) ? fprintf (file, "block output operations mean:%.15f\n", process_usage_medi
  ((process_usage_desviopadrao.ru_oublock_st > 0) ? fprintf (file, "block output operations std deviation:%.15f\n", pr
1480

  (((process_usage.ru_msgsnd > 0) || (process_usage_media.ru_msgsnd_st > 0) || (process_usage_desviopadrao.ru_msgsnd_s
  (((process_usage.ru_msgsnd > 0) || (process_usage_media.ru_msgsnd_st > 0) || (process_usage_desviopadrao.ru_msgsnd_s
  ((process_usage_media.ru_msgsnd_st > 0) ? fprintf (file, "messages sent mean:%.15f\n", process_usage_media.ru_msgsnc
  ((process_usage_desviopadrao.ru_msgsnd_st > 0) ? fprintf (file, "messages sent std deviation:%.15f\n", process_usage

  (((process_usage.ru_msgrcv > 0) || (process_usage_media.ru_msgrcv_st > 0) || (process_usage_desviopadrao.ru_msgrcv_s
  (((process_usage.ru_msgrcv > 0) || (process_usage_media.ru_msgrcv_st > 0) || (process_usage_desviopadrao.ru_msgrcv_s
  ((process_usage_media.ru_msgrcv_st > 0) ? fprintf (file, "messages received mean:%.15f\n", process_usage_media.ru_ms
  ((process_usage_desviopadrao.ru_msgrcv_st > 0) ? fprintf (file, "messages received std deviation:%.15f\n", process_

  (((process_usage.ru_nsignals > 0) || (process_usage_media.ru_nsignals_st > 0) || (process_usage_desviopadrao.ru_nsi
  (((process_usage.ru_nsignals > 0) || (process_usage_media.ru_nsignals_st > 0) || (process_usage_desviopadrao.ru_nsi
  ((process_usage_media.ru_nsignals_st > 0) ? fprintf (file, "signals received mean:%.15f\n", process_usage_media.ru_
  ((process_usage_desviopadrao.ru_nsignals_st > 0) ? fprintf (file, "signals received std deviation:%.15f\n", process

  (((process_usage.ru_nvcsw > 0) || (process_usage_media.ru_nvcsw_st > 0) || (process_usage_desviopadrao.ru_nvcsw_st
  (((process_usage.ru_nvcsw > 0) || (process_usage_media.ru_nvcsw_st > 0) || (process_usage_desviopadrao.ru_nvcsw_st
  ((process_usage_media.ru_nvcsw_st > 0) ? fprintf (file, "voluntary context switches mean:%.15f\n", process_usage_me
  ((process_usage_desviopadrao.ru_nvcsw_st > 0) ? fprintf (file, "voluntary context switches std deviation:%.15f\n",

  (((process_usage.ru_nivcsw > 0) || (process_usage_media.ru_nivcsw_st > 0) || (process_usage_desviopadrao.ru_nivcsw_
  (((process_usage.ru_nivcsw > 0) || (process_usage_media.ru_nivcsw_st > 0) || (process_usage_desviopadrao.ru_nivcsw_
  ((process_usage_media.ru_nivcsw_st > 0) ? fprintf (file, "involuntary context switches mean:%.15f\n", process_usage
  ((process_usage_desviopadrao.ru_nivcsw_st > 0) ? fprintf (file, "involuntary context switches std deviation:%.15f\n

  (((cpu_percentage > 0) || (cpu_percentage_media > 0) || (cpu_percentage_desviopadrao > 0)) ? fprintf (file, "CPU pe
  ((cpu_percentage_media > 0) ? fprintf (file, "CPU percentage mean:%.15f\n", cpu_percentage_media) : 0);
  ((cpu_percentage_desviopadrao > 0) ? fprintf (file, "CPU percentage std deviation:%.15f\n", cpu_percentage_desviopa

  (((sys_call_in_cont > 0) || (sys_call_in_cont_media > 0) || (sys_call_in_cont_desviopadrao > 0)) ? fprintf (file, "
  (((sys_call_in_cont > 0) || (sys_call_in_cont_media > 0) || (sys_call_in_cont_desviopadrao > 0)) ? fprintf (file, "
  ((sys_call_in_cont_media > 0) ? fprintf (file, "number of system calls mean:%.15f\n", sys_call_in_cont_media) : 0);
  ((sys_call_in_cont_desviopadrao > 0) ? fprintf (file, "number of system calls std deviation:%.15f\n", sys_call_in_c

  fflush (file);
  return 0;
1517 }

int
print_proc_stat_summary (element_proc_stat proc_stat, FILE *file, int debug)
{

  fprintf (file, "/proc/PID/stat summary\n");
  fprintf (file, "process virtual memory size:%ubytes\n", proc_stat.vsize);
  fprintf (file, "process resident set size:%upages\n", proc_stat.rss);

  fflush (file);
  return 0;
}

int
print_sys_call_summary (long int sys_call_no[], char *sys_call_name[], double sys_call_no_media[], double sys_call_n
{
```

```
  unsigned int i;

  if (debug)
    fprintf (stderr, "\nprint_sys_call_summary\n");

  fprintf (file, "\nSystem calls summary\n");
  for (i=1; i<NO_SYS_CALLS_PLUSONE; i++){
    (((sys_call_no[i] > 0) || (sys_call_no_media[i]> 0)) ? fprintf (file, "%s:%ld\n",sys_call_name[i], sys_call_no[i])
    (((sys_call_no[i] > 0) || (sys_call_no_media[i]> 0)) ? fprintf (file, "%s difference:%ld\n", sys_call_name[i], sys
    ((sys_call_no_media[i] > 0) ? fprintf (file, "%s mean:%.15f\n", sys_call_name[i], sys_call_no_media[i]) : 0);
    ((sys_call_no_desviopadrao[i] > 0) ? fprintf (file, "%s std deviation:%.15f\n", sys_call_name[i], sys_call_no_desv
  }

  fflush (file);
  return 0;
}


int
print_getrusage_summary_table (unsigned long int indice, double momento_insercao, struct rusage process_usage, struct
1554 {
  double twu, tws, twu_media, tws_media, twu_desviopadrao, tws_desviopadrao, twu_diff, tws_diff;

  if (debug)
    fprintf (stderr, "\nprint_getrusage_summary_table - %u \n", i);

  twu = (((double) process_usage.ru_utime.tv_sec) * M + process_usage.ru_utime.tv_usec);
  twu_media = (((double) process_usage_media.ru_utime_st.tv_sec) * M + process_usage_media.ru_utime_st.tv_usec);
  twu_desviopadrao = (((double) process_usage_desviopadrao.ru_utime_st.tv_sec) * M + process_usage_desviopadrao.ru_ut
  twu_diff = (((double) process_usage_diff.ru_utime.tv_sec) * M + process_usage_diff.ru_utime.tv_usec);
  tws = (((double) process_usage.ru_stime.tv_sec) * M + process_usage.ru_stime.tv_usec);
  tws_media = (((double) process_usage_media.ru_stime_st.tv_sec) * M + process_usage_media.ru_stime_st.tv_usec);
  tws_desviopadrao = (((double) process_usage_desviopadrao.ru_stime_st.tv_sec) * M + process_usage_desviopadrao.ru_st
  tws_diff = (((double) process_usage_diff.ru_stime.tv_sec) * M + process_usage_diff.ru_stime.tv_usec);

  fprintf (file, "%lu  %.15f   ", indice, momento_insercao);

  (((twu > 0) || (twu_media > 0) || (twu_desviopadrao > 0)) ? fprintf (file, "user_time  %.15f  ", twu) : 0);
  (((twu > 0) || (twu_media > 0) || (twu_desviopadrao > 0)) ? fprintf (file, "%.15f  ", twu_diff) : 0);
  ((twu_media > 0) ? fprintf (file, "%.15f  ", twu_media) : 0);
  ((twu_desviopadrao > 0) ? fprintf (file, "%.15f  ", twu_desviopadrao) : 0);

  (((tws > 0) || (tws_media > 0) || (tws_desviopadrao > 0)) ? fprintf (file, "sys_time  %.15f  ", tws) : 0);
  (((tws > 0) || (tws_media > 0) || (twu_desviopadrao > 0)) ? fprintf (file, "%.15f  ", tws_diff) : 0);
  ((tws_media > 0) ? fprintf (file, "%.15f  ", tws_media) : 0);
  ((tws_desviopadrao > 0) ? fprintf (file, "%.15f  ", tws_desviopadrao) : 0);

  (((process_usage.ru_maxrss > 0) || (process_usage_media.ru_maxrss_st > 0) || (process_usage_desviopadrao.ru_maxrss_
  (((process_usage.ru_maxrss > 0) || (process_usage_media.ru_maxrss_st > 0) || (process_usage_desviopadrao.ru_maxrss_
  ((process_usage_media.ru_maxrss_st > 0) ? fprintf (file, "%.15f  ", process_usage_media.ru_maxrss_st) : 0);
  ((process_usage_desviopadrao.ru_maxrss_st > 0) ? fprintf (file, "%.15f  ", process_usage_desviopadrao.ru_maxrss_st)

  (((process_usage.ru_ixrss > 0) || (process_usage_media.ru_ixrss_st > 0) || (process_usage_desviopadrao.ru_ixrss_st
  (((process_usage.ru_ixrss > 0) || (process_usage_media.ru_ixrss_st > 0) || (process_usage_desviopadrao.ru_ixrss_st
  ((process_usage_media.ru_ixrss_st > 0) ? fprintf (file, "%.15f  ", process_usage_media.ru_ixrss_st) : 0);
  ((process_usage_desviopadrao.ru_ixrss_st > 0) ? fprintf (file, "%.15f  ", process_usage_desviopadrao.ru_ixrss_st) :

1591 (((process_usage.ru_idrss > 0) || (process_usage_media.ru_idrss_st > 0) || (process_usage_desviopadrao.ru_idrss_st
  (((process_usage.ru_idrss > 0) || (process_usage_media.ru_idrss_st > 0) || (process_usage_desviopadrao.ru_idrss_st
  ((process_usage_media.ru_idrss_st > 0) ? fprintf (file, "%.15f  ", process_usage_media.ru_idrss_st) : 0);
  ((process_usage_desviopadrao.ru_idrss_st > 0) ? fprintf (file, "%.15f  ", process_usage_desviopadrao.ru_idrss_st) :

  (((process_usage.ru_isrss > 0) || (process_usage_media.ru_isrss_st > 0) || (process_usage_desviopadrao.ru_isrss_st
  (((process_usage.ru_isrss > 0) || (process_usage_media.ru_isrss_st > 0) || (process_usage_desviopadrao.ru_isrss_st
  ((process_usage_media.ru_isrss_st > 0) ? fprintf (file, "%.15f  ", process_usage_media.ru_isrss_st) : 0);
  ((process_usage_desviopadrao.ru_isrss_st > 0) ? fprintf (file, "%.15f  ", process_usage_desviopadrao.ru_isrss_st) :

  (((process_usage.ru_minflt > 0) || (process_usage_media.ru_minflt_st > 0) || (process_usage_desviopadrao.ru_minflt_
  (((process_usage.ru_minflt > 0) || (process_usage_media.ru_minflt_st > 0) || (process_usage_desviopadrao.ru_minflt_
  ((process_usage_media.ru_minflt_st > 0) ? fprintf (file, "%.15f  ", process_usage_media.ru_minflt_st) : 0);
  ((process_usage_desviopadrao.ru_minflt_st > 0) ? fprintf (file, "%.15f  ", process_usage_desviopadrao.ru_minflt_st)

  (((process_usage.ru_majflt > 0) || (process_usage_media.ru_majflt_st > 0) || (process_usage_desviopadrao.ru_majflt_
```

```
  (((process_usage.ru_majflt > 0) || (process_usage_media.ru_majflt_st > 0) || (process_usage_desviopadrao.ru_majflt_s
  ((process_usage_media.ru_majflt_st > 0) ? fprintf (file, "%.15f  ", process_usage_media.ru_majflt_st) : 0);
  ((process_usage_desviopadrao.ru_majflt_st > 0) ? fprintf (file, "%.15f  ", process_usage_desviopadrao.ru_majflt_st)

  (((process_usage.ru_nswap > 0) || (process_usage_media.ru_nswap_st > 0) || (process_usage_desviopadrao.ru_nswap_st >
  (((process_usage.ru_nswap > 0) || (process_usage_media.ru_nswap_st > 0) || (process_usage_desviopadrao.ru_nswap_st >
  ((process_usage_media.ru_nswap_st > 0) ? fprintf (file, "%.15f  ", process_usage_media.ru_nswap_st) : 0);
  ((process_usage_desviopadrao.ru_nswap_st > 0) ? fprintf (file, "%.15f  ", process_usage_desviopadrao.ru_nswap_st) :

  (((process_usage.ru_inblock > 0) || (process_usage_media.ru_inblock_st > 0) || (process_usage_desviopadrao.ru_inbloc
  (((process_usage.ru_inblock > 0) || (process_usage_media.ru_inblock_st > 0) || (process_usage_desviopadrao.ru_inbloc
  ((process_usage_media.ru_inblock_st > 0) ? fprintf (file, "%.15f  ", process_usage_media.ru_inblock_st) : 0);
  ((process_usage_desviopadrao.ru_inblock_st > 0) ? fprintf (file, "%.15f  ", process_usage_desviopadrao.ru_inblock_st

  (((process_usage.ru_oublock > 0) || (process_usage_media.ru_oublock_st > 0) || (process_usage_desviopadrao.ru_oubloc
  (((process_usage.ru_oublock > 0) || (process_usage_media.ru_oublock_st > 0) || (process_usage_desviopadrao.ru_oubloc
  ((process_usage_media.ru_oublock_st > 0) ? fprintf (file, "%.15f  ", process_usage_media.ru_oublock_st) : 0);
  ((process_usage_desviopadrao.ru_oublock_st > 0) ? fprintf (file, "%.15f  ", process_usage_desviopadrao.ru_oublock_st

  (((process_usage.ru_msgsnd > 0) || (process_usage_media.ru_msgsnd_st > 0) || (process_usage_desviopadrao.ru_msgsnd_s
  (((process_usage.ru_msgsnd > 0) || (process_usage_media.ru_msgsnd_st > 0) || (process_usage_desviopadrao.ru_msgsnd_s
1628 ((process_usage_media.ru_msgsnd_st > 0) ? fprintf (file, "%.15f  ", process_usage_media.ru_msgsnd_st) : 0);
  ((process_usage_desviopadrao.ru_msgsnd_st > 0) ? fprintf (file, "%.15f  ", process_usage_desviopadrao.ru_msgsnd_st)

  (((process_usage.ru_msgrcv > 0) || (process_usage_media.ru_msgrcv_st > 0) || (process_usage_desviopadrao.ru_msgrcv_s
  (((process_usage.ru_msgrcv > 0) || (process_usage_media.ru_msgrcv_st > 0) || (process_usage_desviopadrao.ru_msgrcv_s
  ((process_usage_media.ru_msgrcv_st > 0) ? fprintf (file, "%.15f  ", process_usage_media.ru_msgrcv_st) : 0);
  ((process_usage_desviopadrao.ru_msgrcv_st > 0) ? fprintf (file, "%.15f  ", process_usage_desviopadrao.ru_msgrcv_st)

  (((process_usage.ru_nsignals > 0) || (process_usage_media.ru_nsignals_st > 0) || (process_usage_desviopadrao.ru_nsi
  (((process_usage.ru_nsignals > 0) || (process_usage_media.ru_nsignals_st > 0) || (process_usage_desviopadrao.ru_nsi
  ((process_usage_media.ru_nsignals_st > 0) ? fprintf (file, "%.15f  ", process_usage_media.ru_nsignals_st) : 0);
  ((process_usage_desviopadrao.ru_nsignals_st > 0) ? fprintf (file, "%.15f  ", process_usage_desviopadrao.ru_nsignals

  (((process_usage.ru_nvcsw > 0) || (process_usage_media.ru_nvcsw_st > 0) || (process_usage_desviopadrao.ru_nvcsw_st
  (((process_usage.ru_nvcsw > 0) || (process_usage_media.ru_nvcsw_st > 0) || (process_usage_desviopadrao.ru_nvcsw_st
  ((process_usage_media.ru_nvcsw_st > 0) ? fprintf (file, "%.15f  ", process_usage_media.ru_nvcsw_st) : 0);
  ((process_usage_desviopadrao.ru_nvcsw_st > 0) ? fprintf (file, "%.15f  ", process_usage_desviopadrao.ru_nvcsw_st) :

  (((process_usage.ru_nivcsw > 0) || (process_usage_media.ru_nivcsw_st > 0) || (process_usage_desviopadrao.ru_nivcsw_
  (((process_usage.ru_nivcsw > 0) || (process_usage_media.ru_nivcsw_st > 0) || (process_usage_desviopadrao.ru_nivcsw_
  ((process_usage_media.ru_nivcsw_st > 0) ? fprintf (file, "%.15f  ", process_usage_media.ru_nivcsw_st) : 0);
  ((process_usage_desviopadrao.ru_nivcsw_st > 0) ? fprintf (file, "%.15f  ", process_usage_desviopadrao.ru_nivcsw_st)

  (((cpu_percentage > 0) || (cpu_percentage_media > 0) || (cpu_percentage_desviopadrao > 0)) ? fprintf (file, "cpu_p
  ((cpu_percentage_media > 0) ? fprintf (file, "%.15f  ", cpu_percentage_media) : 0);
  ((cpu_percentage_desviopadrao > 0) ? fprintf (file, "%.15f  ", cpu_percentage_desviopadrao) : 0);

  (((sys_call_in_cont > 0) || (sys_call_in_cont_media > 0) || (sys_call_in_cont_desviopadrao > 0)) ? fprintf (file,
  (((sys_call_in_cont > 0) || (sys_call_in_cont_media > 0) || (sys_call_in_cont_desviopadrao > 0)) ? fprintf (file,
  ((sys_call_in_cont_media > 0) ? fprintf (file, "%.15f  ", sys_call_in_cont_media) : 0);
  ((sys_call_in_cont_desviopadrao > 0) ? fprintf (file, "%.15f  ", sys_call_in_cont_desviopadrao) : 0);

  fflush (file);
  return 0;
}

int
1665 print_sys_call_summary_table (unsigned long int indice, double momento_insercao, long int sys_call_no[], char *sys_c
{
  unsigned int i;

  if (debug)
    fprintf (stderr, "\nprint_sys_call_summary_table\n");

  fprintf (file, "%lu  %.15f   ", indice, momento_insercao);
  for (i=1; i<NO_SYS_CALLS_PLUSONE; i++){
    (((sys_call_no[i] > 0) || (sys_call_no_media[i]> 0)) ? fprintf (file, "%s  %ld  ", sys_call_name[i], sys_call_nc
    (((sys_call_no[i] > 0) || (sys_call_no_media[i]> 0)) ? fprintf (file, "%ld  ", sys_call_no_diff[i]) : 0);
    ((sys_call_no_media[i] > 0) ? fprintf (file, "%.15f  ", sys_call_no_media[i]) : 0);
    ((sys_call_no_desviopadrao[i] > 0) ? fprintf (file, "%.15f  ", sys_call_no_desviopadrao[i]) : 0);
  }
```

```
  fflush (file);
  return 0;
}

int
gen_proc_summary (struct timeval utime, struct timeval stime, struct timeval endtrace, int starttime, double *process_
{
  /* reading of /proc/stat */
  int status = 0;
  int fscanf_error;
  unsigned long int btime;
  FILE *proc_system_statf;
  char *proc_system_stat = NULL;
  char tmpstring[MAXPATHLEN];

  /*
     Open /proc/stat file
  */
  if (strlen("/proc/stat") < MAXPATHLEN){
    proc_system_stat = strdup ("/proc/stat");
    if ((proc_system_statf = fopen(proc_system_stat, "r")) == NULL){
      fprintf(stderr, "gen_proc_summary - can't fopen '%s': %s\n", proc_system_stat, strerror(errno));
1702  status = -1;
    }
  }

  /*
     Reading /proc/stat
  */
  memset(tmpstring, '\0', sizeof (tmpstring));
  if (proc_system_statf != NULL){
    if (fseek (proc_system_statf, 0, SEEK_SET) < 0){
      fprintf (stderr, "gen_proc_summary - fseek error on file: %s.\n", proc_system_stat);
      status = -1;
    } else {
      while (strncmp (tmpstring, "btime", strlen("btime")) != 0){
        if (((fscanf_error = fscanf (proc_system_statf, "%s %lu", tmpstring, &btime)) == 0) || (fscanf_error == EOF)){
          fprintf (stderr, "gen_proc_summary - fscanf error on file: %s.\n", proc_system_stat);
          status = -1;
          break;
        }
      }
    }
  }

  /*
     closing /proc/stat
  */

  if (proc_system_statf != NULL){
    if ((fclose (proc_system_statf))==EOF){
      fprintf(stderr, "gen_proc_summary - can't fclose '%s': %s\n", proc_system_stat, strerror(errno));
    }
  }

  if (status == 0){
    /* Calculating process run time, %CPU usage */

    if (timeval_add (effective_run_time, utime, stime, debug) != 0){
1739  perror ("gen_proc_summary - timeval_add error");
      status = -1;
    }

    *process_start_time = btime + (((double) starttime) / 100);
    *real_run_time = (((endtrace.tv_sec + ((double) endtrace.tv_usec) / M) < *process_start_time) ? (effective_run_ti
  }

  if (debug){
    fprintf (stderr, "\ngen_proc_summary - process_start_time:%1.15fs\n", *process_start_time);
    fprintf (stderr, "gen_proc_summary - endtrace:%1.15fs\n", (endtrace.tv_sec + ((double) endtrace.tv_usec) / M));
    fprintf (stderr, "gen_proc_summary - real_run_time:%1.15fs\n", *real_run_time);
  }
  return status;
```

```
}

int
sys_call_name_init (char *sys_call_name[])
{

  sys_call_name[1] = strdup ("exit");
  sys_call_name[2] = strdup ("fork");
  sys_call_name[3] = strdup ("read");
  sys_call_name[4] = strdup ("write");
  sys_call_name[5] = strdup ("open");
  sys_call_name[6] = strdup ("close");
  sys_call_name[7] = strdup ("waitpid");
  sys_call_name[8] = strdup ("creat");
  sys_call_name[9] = strdup ("link");
  sys_call_name[10] = strdup ("unlink");
  sys_call_name[11] = strdup ("execve");
  sys_call_name[12] = strdup ("chdir");
  sys_call_name[13] = strdup ("time");
  sys_call_name[14] = strdup ("mknod");
  sys_call_name[15] = strdup ("chmod");
  sys_call_name[16] = strdup ("lchown");
  sys_call_name[17] = strdup ("break");
1776  sys_call_name[18] = strdup ("oldstat");
  sys_call_name[19] = strdup ("lseek");
  sys_call_name[20] = strdup ("getpid");
  sys_call_name[21] = strdup ("mount");
  sys_call_name[22] = strdup ("umount");
  sys_call_name[23] = strdup ("setuid");
  sys_call_name[24] = strdup ("getuid");
  sys_call_name[25] = strdup ("stime");
  sys_call_name[26] = strdup ("ptrace");
  sys_call_name[27] = strdup ("alarm");
  sys_call_name[28] = strdup ("oldfstat");
  sys_call_name[29] = strdup ("pause");
  sys_call_name[30] = strdup ("utime");
  sys_call_name[31] = strdup ("stty");
  sys_call_name[32] = strdup ("gtty");
  sys_call_name[33] = strdup ("access");
  sys_call_name[34] = strdup ("nice");
  sys_call_name[35] = strdup ("ftime");
  sys_call_name[36] = strdup ("sync");
  sys_call_name[37] = strdup ("kill");
  sys_call_name[38] = strdup ("rename");
  sys_call_name[39] = strdup ("mkdir");
  sys_call_name[40] = strdup ("rmdir");
  sys_call_name[41] = strdup ("dup");
  sys_call_name[42] = strdup ("pipe");
  sys_call_name[43] = strdup ("times");
  sys_call_name[44] = strdup ("prof");
  sys_call_name[45] = strdup ("brk");
  sys_call_name[46] = strdup ("setgid");
  sys_call_name[47] = strdup ("getgid");
  sys_call_name[48] = strdup ("signal");
  sys_call_name[49] = strdup ("geteuid");
  sys_call_name[50] = strdup ("getegid");
  sys_call_name[51] = strdup ("acct");
  sys_call_name[52] = strdup ("umount2");
  sys_call_name[53] = strdup ("lock");
  sys_call_name[54] = strdup ("ioctl");
1813  sys_call_name[55] = strdup ("fcntl");
  sys_call_name[56] = strdup ("mpx");
  sys_call_name[57] = strdup ("setpgid");
  sys_call_name[58] = strdup ("ulimit");
  sys_call_name[59] = strdup ("oldolduname");
  sys_call_name[60] = strdup ("umask");
  sys_call_name[61] = strdup ("chroot");
  sys_call_name[62] = strdup ("ustat");
  sys_call_name[63] = strdup ("dup2");
  sys_call_name[64] = strdup ("getppid");
  sys_call_name[65] = strdup ("getpgrp");
  sys_call_name[66] = strdup ("setsid");
  sys_call_name[67] = strdup ("sigaction");
```

```
sys_call_name[68] = strdup ("sgetmask");
sys_call_name[69] = strdup ("ssetmask");
sys_call_name[70] = strdup ("setreuid");
sys_call_name[71] = strdup ("setregid");
sys_call_name[72] = strdup ("sigsuspend");
sys_call_name[73] = strdup ("sigpending");
sys_call_name[74] = strdup ("sethostname");
sys_call_name[75] = strdup ("setrlimit");
sys_call_name[76] = strdup ("getrlimit");
sys_call_name[77] = strdup ("getrusage");
sys_call_name[78] = strdup ("gettimeofday");
sys_call_name[79] = strdup ("settimeofday");
sys_call_name[80] = strdup ("getgroups");
sys_call_name[81] = strdup ("setgroups");
sys_call_name[82] = strdup ("select");
sys_call_name[83] = strdup ("symlink");
sys_call_name[84] = strdup ("oldlstat");
sys_call_name[85] = strdup ("readlink");
sys_call_name[86] = strdup ("uselib");
sys_call_name[87] = strdup ("swapon");
sys_call_name[88] = strdup ("reboot");
sys_call_name[89] = strdup ("readdir");
sys_call_name[90] = strdup ("mmap");
sys_call_name[91] = strdup ("munmap");
sys_call_name[92] = strdup ("truncate");
sys_call_name[93] = strdup ("ftruncate");
sys_call_name[94] = strdup ("fchmod");
sys_call_name[95] = strdup ("fchown");
sys_call_name[96] = strdup ("getpriority");
sys_call_name[97] = strdup ("setpriority");
sys_call_name[98] = strdup ("profil");
sys_call_name[99] = strdup ("statfs");
sys_call_name[100] = strdup ("fstatfs");
sys_call_name[101] = strdup ("ioperm");
sys_call_name[102] = strdup ("socketcall");
sys_call_name[103] = strdup ("syslog");
sys_call_name[104] = strdup ("setitimer");
sys_call_name[105] = strdup ("getitimer");
sys_call_name[106] = strdup ("stat");
sys_call_name[107] = strdup ("lstat");
sys_call_name[108] = strdup ("fstat");
sys_call_name[109] = strdup ("olduname");
sys_call_name[110] = strdup ("iopl");
sys_call_name[111] = strdup ("vhangup");
sys_call_name[112] = strdup ("idle");
sys_call_name[113] = strdup ("vm86old");
sys_call_name[114] = strdup ("wait4");
sys_call_name[115] = strdup ("swapoff");
sys_call_name[116] = strdup ("sysinfo");
sys_call_name[117] = strdup ("ipc");
sys_call_name[118] = strdup ("fsync");
sys_call_name[119] = strdup ("sigreturn");
sys_call_name[120] = strdup ("clone");
sys_call_name[121] = strdup ("setdomainname");
sys_call_name[122] = strdup ("uname");
sys_call_name[123] = strdup ("modify_ldt");
sys_call_name[124] = strdup ("adjtimex");
sys_call_name[125] = strdup ("mprotect");
sys_call_name[126] = strdup ("sigprocmask");
sys_call_name[127] = strdup ("create_module");
sys_call_name[128] = strdup ("init_module");
sys_call_name[129] = strdup ("delete_module");
sys_call_name[130] = strdup ("get_kernel_syms");
sys_call_name[131] = strdup ("quotactl");
sys_call_name[132] = strdup ("getpgid");
sys_call_name[133] = strdup ("fchdir");
sys_call_name[134] = strdup ("bdflush");
sys_call_name[135] = strdup ("sysfs");
sys_call_name[136] = strdup ("personality");
sys_call_name[137] = strdup ("afs_syscall");
sys_call_name[138] = strdup ("setfsuid");
sys_call_name[139] = strdup ("setfsgid");
sys_call_name[140] = strdup ("_llseek");
```

```
        sys_call_name[141] = strdup ("getdents");
        sys_call_name[142] = strdup ("_newselect");
        sys_call_name[143] = strdup ("flock");
        sys_call_name[144] = strdup ("msync");
        sys_call_name[145] = strdup ("readv");
        sys_call_name[146] = strdup ("writev");
        sys_call_name[147] = strdup ("getsid");
        sys_call_name[148] = strdup ("fdatasync");
        sys_call_name[149] = strdup ("_sysctl");
        sys_call_name[150] = strdup ("mlock");
        sys_call_name[151] = strdup ("munlock");
        sys_call_name[152] = strdup ("mlockall");
        sys_call_name[153] = strdup ("munlockall");
        sys_call_name[154] = strdup ("sched_setparam");
        sys_call_name[155] = strdup ("sched_getparam");
        sys_call_name[156] = strdup ("sched_setscheduler");
        sys_call_name[157] = strdup ("sched_getschedule");
        sys_call_name[158] = strdup ("sched_yield");
        sys_call_name[159] = strdup ("sched_get_priority_max");
        sys_call_name[160] = strdup ("sched_get_priority_min");
        sys_call_name[161] = strdup ("sched_rr_get_interval");
        sys_call_name[162] = strdup ("nanosleep");
        sys_call_name[163] = strdup ("mremap");
        sys_call_name[164] = strdup ("setresuid");
        sys_call_name[165] = strdup ("getresuid");
1924    sys_call_name[166] = strdup ("vm86");
        sys_call_name[167] = strdup ("query_module");
        sys_call_name[168] = strdup ("poll");
        sys_call_name[169] = strdup ("nfsservctl");
        sys_call_name[170] = strdup ("setresgid");
        sys_call_name[171] = strdup ("getresgid");
        sys_call_name[172] = strdup ("prctl");
        sys_call_name[173] = strdup ("rt_sigreturn");
        sys_call_name[174] = strdup ("rt_sigaction");
        sys_call_name[175] = strdup ("rt_sigprocmask");
        sys_call_name[176] = strdup ("rt_sigpending");
        sys_call_name[177] = strdup ("rt_sigtimedwait");
        sys_call_name[178] = strdup ("rt_sigqueueinfo");
        sys_call_name[179] = strdup ("rt_sigsuspend");
        sys_call_name[180] = strdup ("pread");
        sys_call_name[181] = strdup ("pwrite");
        sys_call_name[182] = strdup ("chown");
        sys_call_name[183] = strdup ("getcwd");
        sys_call_name[184] = strdup ("capget");
        sys_call_name[185] = strdup ("capset");
        sys_call_name[186] = strdup ("sigaltstack");
        sys_call_name[187] = strdup ("sendfile");
        sys_call_name[188] = strdup ("getpmsg");
        sys_call_name[189] = strdup ("putpmsg");
        sys_call_name[190] = strdup ("vfork");
        sys_call_name[191] = strdup ("ugetrlimit");
        sys_call_name[192] = strdup ("mmap2");
        sys_call_name[193] = strdup ("truncate64");
        sys_call_name[194] = strdup ("ftruncate64");
        sys_call_name[195] = strdup ("stat64");
        sys_call_name[196] = strdup ("lstat64");
        sys_call_name[197] = strdup ("fstat64");
        sys_call_name[198] = strdup ("lchown32");
        sys_call_name[199] = strdup ("getuid32");
        sys_call_name[200] = strdup ("getgid32");
        sys_call_name[201] = strdup ("geteuid32");
        sys_call_name[202] = strdup ("getegid32");
1961    sys_call_name[203] = strdup ("setreuid32");
        sys_call_name[204] = strdup ("setregid32");
        sys_call_name[205] = strdup ("getgroups32");
        sys_call_name[206] = strdup ("setgroups32");
        sys_call_name[207] = strdup ("fchown32");
        sys_call_name[208] = strdup ("setresuid32");
        sys_call_name[209] = strdup ("getresuid32");
        sys_call_name[210] = strdup ("setresgid32");
        sys_call_name[211] = strdup ("getresgid32");
        sys_call_name[212] = strdup ("chown32");
        sys_call_name[213] = strdup ("setuid32");
```

```
  sys_call_name[214] = strdup ("setgid32");
  sys_call_name[215] = strdup ("setfsuid32");
  sys_call_name[216] = strdup ("setfsgid32");
  sys_call_name[217] = strdup ("pivot_root");
  sys_call_name[218] = strdup ("mincore");
  sys_call_name[219] = strdup ("madvise");
  sys_call_name[220] = strdup ("getdents64");
  sys_call_name[221] = strdup ("fcntl64");
  return 0;
}
```

REFERÊNCIAS BIBLIOGRÁFICAS

[1] Wichert Akkerman. *LINUX MAN - STRACE(1).* Debian, 1999.

[2] Maurice J. Bach. *THE DESIGN OF THE UNIX OPERATING SYSTEM.* Prentice Hall, Inc, 1987.

[3] Paul Robert Barford. *MODELING, MEASUREMENT AND PERFORMANCE OF WORLD WIDE WEB TRANSACTIONS.* Tese de Doutorado, Boston University - Graduate School of Arts and Sciences, 2001.

[4] Ivan Bowman. Conceptual architecture of the linux kernel. Relatório técnico, University of Waterloo, 1998.

[5] D. D. Clark e D. L. Tennenhouse. Architectural considerations for a new generation of protocols. *ACMSIGCOMM'90*, 1990.

[6] Chris Dalton, Greg Watson, David Banks, Costas Calamvokis, Aled Edwards, e John Lumley. Afterburner. *IEEE Network*, july de 1993.

[7] Jussara Marques de Almeida. Investigação sobre a interacao entre um sistema operacional e um servidor web. Dissertação de Mestrado, DCC - Universidade Federal de Minas Gerais, 1997.

[8] Peter Druschel. Operating system support for high-speed communication. *Communications of the ACM*, september de 1996.

[9] Peter Druschel, Mark B. Abbot, Michael A. Pagels, e Larry L. Peterson. Network subsystem design. *IEEE Network*, july de 1993.

[10] John L. Henessy e David A. Patterson. *COMPUTER ARCHITECTURE A QUANTITATIVE APPROACH.* Morgan Kaufman Publishers, 1996.

[11] Yiming Hu, Ashwini Nanda, e Qing Yang. Measurement, analysis and performance improvement of the apache web server. *18th IEEE International Performance Computing and Communications Conference*, 1999.

[12] Raj Jain. *THE ART OF COMPUTER SYSTEMS PERFORMANCE ANALYSIS: TECHNIQUES FOR EXPERIMENTAL DESIGN, MEASUREMENT, SIMULATION AND MODELING.* John Wiley & Sons, Inc., 1991.

[13] Michael K. Johnson. *LINUX MAN - PS(1).* RedHat, 1997.

[14] Gerlof Langeveld. *LINUX MAN - ATSAR(1).* AT Computing, 2001.

[15] Edward D. Lazowska, John Zahorjan, G. Scott Graham, e Kenneth C. Sevcik. *QUANTITATIVE SYSTEM PERFORMANCE: COMPUTER SYSTEM ANALYSIS USING QUEUEING NETWORK MODELS.* Prentice-Hall, Inc., 1984.

[16] Richard P. Martin e David E. Culler. NFS sensitivity to high performance networks. *SIGMETRICS'99.*

[17] Jean Paul Meynard. Study of network performance analysis tools in simulation including real-time constraints. Relatório técnico, Linkopings Universitet - Department of Computer and Information Science, 1997.

[18] J. Mogul. The case for persistent-connection HTTP. Relatório técnico, DEC Western Research Laboratory, Palo Alto, CA USA, 1995.

[19] Joseph Pasquale, Eric Anderson, e P. Keith Muller. Container shipping: Operating system support for i/o-intensive applications. *IEEE Computer*, 1994.

[20] David Robinson e the Apache Group. *APACHE - AN HTTP SERVER, REFERENCE MANUAL*, 1995.

[21] David A. Rusling. The linux kernel, version 0.8-3.

[22] Michael Shields. *LINUX MAN - TOP(1).* University of Denver - Department of Computer Science, 1993.

[23] Jonathan M. Smith e C. Brendan S. Traw. Giving applications access to gb/s networking. *IEEE Network*, july de 1993.

[24] Andrew S. Tanenbaum. *MODERN OPERATING SYSTEMS.* Prentice Hall, Inc, 2001.

[25] Linus Torvalds. *LINUX MAN - PROC(5).* Transmeta, 1996.

[26] Linus Torvalds. *LINUX MAN - GETRLIMIT(2).* Transmeta, 2001.

[27] M. Welsh. *THE LINUX BIBLE.* Yggdrasil Computing, Inc, 1994.