UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIAS DA COMPUTAÇÃO – BACHARELADO

FERRAMENTA ADMINISTRATIVA PARA AMBIENTES COMPUTACIONAIS

LIANDRO ROSSINI

BLUMENAU
2004

2004/2-33

LIANDRO ROSSINI

FERRAMENTA ADMINISTRATIVA PARA AMBIENTES COMPUTACIONAIS

Trabalho de Conclusão de Curso submetido à Universidade Regional de Blumenau para a obtenção dos créditos na disciplina Trabalho de Conclusão de Curso II do curso de Ciência da Computação — Bacharelado.

Prof. Antônio Carlos Tavares - Orientador

BLUMENAU
2004

2004/2-33

FERRAMENTA ADMINISTRATIVA PARA AMBIENTES COMPUTACIONAIS

Por

LIANDRO ROSSINI

Trabalho aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II, pela banca examinadora formada por:

Presidente:	Prof. Antônio Carlos Tavares – Orientador, FURB
Membro:	Prof. Mauro Marcelo Mattos, FURB
Membro:	Prof. Evaristo Baptista, FURB

Blumenau, 17 de Novembro de 2004

Dedico este trabalho à minha mãe, à minha irmã e à minha namorada.

Nada é por acaso.

(Ditado popular)

AGRADECIMENTOS

Agradeço a todos que me incentivaram a concluir este trabalho: minha mãe, minha irmã, minha namorada, e ao meu orientador Antônio Carlos Tavares pelo auxílio e compreensão.

RESUMO

Este trabalho consiste no desenvolvimento de uma ferramenta administrativa para ambientes computacionais. A observação e o acompanhamento na administração de ambientes de médio e grande porte evidenciaram problemas como: falta de espaço em disco, aplicações ativas porém inoperantes, processos inativos, processos ativos e apresentando erros, que podem ser prevenidos/solucionados com esta ferramenta. O sistema aqui apresentado foi desenvolvido para ser executado em um ambiente LINUX e pode ser portado para sistemas operacionais similares. O presente trabalho tem como finalidade auxiliar administradores de ambientes computacionais com suas tarefas, realizando as habituais onde as soluções são conhecidas e prevenindo/solucionando problemas pontuais.

Palavras chaves: Escalonador; Processos; Ambientes Computacionais.

ABSTRACT

This work consists on the development of a tool for administration on computational environments. The monitoring and management of medium and large network and stand-alone computer environments have shown problems like: diskspace shortage, hung active applications, inactive processes, error-prompting active applications, etc. This problems can be solved by this tool. The system shown has been developed for the LINUX platform but may be migrated to similar OSs. The present work goal is to aid network administrators on their tasks and even helping them avoid certain problems.

.

Key-Words: Scheduler; Processes; Computational Ambients.

LISTA DE ILUSTRAÇÕES

SUMÁRIO

1 INTRODUÇÃO

O avanço da tecnologia implica em ajustes constantes no ambiente operacional de todas as empresas. Segundo Davis (1990, p. 34) *"a renovação é, talvez, a única constante no campo da computação"*. Novas soluções são apresentadas para melhorar o funcionamento dos processos e estas soluções refletem em forma de ajustes em vários outros eventos diretamente ou indiretamente relacionados com este primeiro.

A otimização dos processos, o ganho em performance e um maior aproveitamento dos recursos são o ponto mais atacado hoje em dia. No ambiente computacional isto não é diferente. Os recursos que geram maior estudo nesta área são: rede, discos utilizados para o sistema de armazenamento de arquivos, memória, processadores e também o sistema operacional utilizado. Procura-se aproveitar ao máximo a capacidade de um recurso tomando os devidos cuidados para que esta utilização não se reflita em perda de performance. Quanto mais puder se extrair deste recurso e quanto maior o tempo em que este recurso estiver sendo utilizado, melhor. Sabendo que todos os recursos computacionais têm um controle efetuado via sistema operacional, a utilização de alguma ferramenta para o auxílio na gerência deste recurso é a que estará sendo pesquisada.

Pode-se levantar várias questões que causam perdas para uma empresa devido ao fato dos ambientes computacionais estarem sendo mal administrados. Segundo Christian (1987, p. 251) *"a manutenção da integridade do sistema é provavelmente a atividade mais desafiadora para um gerenciador de sistemas"*. Avaliando médias e grandes empresas, a seguinte situação é encontrada: vários servidores com bancos de dados para serem mantidos, servidores *web* com aplicações ativas, servidores de *e-mail*, servidor com aplicações para controle interno (contendo bancos de dados e sistemas ativos).

Várias pessoas deveriam ser envolvidas para monitorar um ambiente desta amplitude, com uma lista de tarefas e itens a serem verificados em cada um dos servidores e bancos para garantir que nada aconteça e que todo este ambiente continue funcionando. Sabe-se porém, que poucas empresas trabalham pró-ativamente com relação a isto. Esperam os erros acontecerem e depois correm atrás dos acertos.

Aqui pode-se dimensionar o valor das ferramentas de auxílio em monitoração de ambientes. Elas são configuradas da forma necessária e fazem este trabalho.

Segundo Hansen (1995, p. 03), *"um sistema operacional provê uma série de serviços básicos para controlar a confusão de detalhes das tarefas, carga de programas na memória e execução dos mesmos, leitura e gravação de dados, segurança, disponibilidade de recursos, comunicações com outros sistemas e dispositivos remotos"*. Os sistemas operacionais por si só, não fazem absolutamente nada quando algumas situações ocorrem no meio onde eles estão trabalhando, situações estas que podem prejudicar até o funcionamento deste próprio sistema operacional. Dentre estas situações, seguem algumas das mais críticas: falta de espaço em estruturas, indisponibilidade de serviços, consumo exagerado de recursos computacionais por algum software que esteja fora de controle, entre outros.

Devido ao conhecimento deste problema, surgiu a proposta para o desenvolvimento de uma ferramenta que faça um controle destas situações. A ferramenta permitirá que sejam cadastradas situações que devem ser monitoradas e para cada uma delas permitirá também que se associe uma ação que deve ser tomada pela própria ferramenta a fim de corrigir erros em ambientes computacionais ou pelo menos diminuir as conseqüências destas situações indesejáveis. Utilizando como exemplo um dos problemas existentes já citados acima, a ferramenta pode ser configurada para monitorar o espaço livre/ocupado em determinada estrutura de um ambiente computacional, assim que este espaço estiver perto de ser utilizado por completo, antes de ficar cheio, a ferramenta pode apagar alguns arquivos pré-determinados ou até mesmo enviar um *e-mail* à algum administrador deste ambiente para que tome as devidas providências.

Uma outra funcionalidade da ferramenta será monitorar *logs* gerados por aplicações e através da monitoração destes *logs* disparar eventos preventivos ou corretivos sobre o ambiente computacional. Uma das monitorações que será feita nestes *logs* é buscar por uma informação conhecida e executar algum processo. A outra monitoração é verificar a data da última atualização de arquivos. Arquivos de *log* que não são atualizados há muito tempo podem identificar um problema na aplicação do usuário.

A ferramenta desenvolvida permitirá ao usuário fazer associações entre os comandos do sistema operacional e os deixar agendados para que mantenham suas aplicações funcionando como, por exemplo: um sistema de alarme atualiza um arquivo de *log* a cada 30 segundos; caso ele pare de atualizar este arquivo por mais de 3 minutos, a ferramenta deve então encerrar este processo e iniciá-lo novamente, pois, provavelmente, ele está com algum problema. Para isto o usuário deverá cadastrar o arquivo de *log* que será monitorado, o tempo crítico (tempo em que a ferramenta deve tomar alguma ação) e também a ação a ser tomada pela ferramenta ao identificar que chegou neste tempo crítico.

1.1 OBJETIVOS DO TRABALHO

O objetivo deste trabalho é desenvolver uma ferramenta que auxilie na administração de ambientes computacionais, permitindo que sejam cadastradas situações a serem monitoradas e então associar a estas situações eventos corretivos ou preventivos.

Os objetivos específicos do trabalho são:

a) gerar um módulo para cadastrar os tipos de problemas mais corriqueiros do ambiente (como falta de espaço, *logs* não atualizados no prazo esperado, processos inativos, processos ativos porém inoperantes);
b) disponibilizar um módulo para que sejam cadastradas e parametrizadas as situações a serem monitoradas pela ferramenta, para cada tipo de problema;
c) desenvolver um escalonador que ficará lendo as informações cadastradas acima, e atuando sobre cada situação.

1.2 ESTRUTURA DO TRABALHO

No primeiro capítulo deste trabalho tem-se a introdução, apresentação dos objetivos e a organização do mesmo.

No segundo capítulo é iniciada a fundamentação teórica com uma introdução a sistemas operacionais, uma descrição sobre os sistemas UNIX e LINUX, e uma breve explicação sobre escalonadores.

O terceiro capítulo é dedicado a administração de ambientes computacionais. Nele são descritas as principais tarefas de um administrador de sistemas com uma breve descrição para cada uma delas.

O quarto capítulo mostra o desenvolvimento do trabalho. Aqui encontra-se a explicação dos principais requisitos e a sua especificação. São abordados também os temas como implementação, técnicas e ferramentas utilizadas e operacionalidades da implementação. No final do terceiro capítulo é apresentada a operacionalidade da ferramenta desenvolvida.

O quinto capítulo foi destinado a um estudo de caso. Nele são apresentadas explicações dos testes realizados assim como as telas colhidas no momento dos testes.

No sexto capítulo temos as conclusões finais e extensões para novos trabalhos.

2 SISTEMAS OPERACIONAIS

Segundo Tanenbaum (2000, p. 17) “Sem *software*, um computador é basicamente um inútil amontoado de metal”. Esta frase expressa a realidade do ambiente computacional, onde uma máquina sem um programa específico de computador para a controlar, não é capaz de realizar qualquer atividade.

Conforme Tanenbaum (2000, p. 17) “O programa de sistema mais fundamental é o sistema operacional, que controla todos os recursos do computador e fornece a base sobre a qual os programas aplicativos podem ser escritos.“ Sem um sistema operacional, ficaria extremamente difícil escrever programas para realizarem tarefas em um computador. Os programadores teriam que se preocupar em cada programa, com uma forma de estar controlando os recursos do computador e isto tornaria os programas redundantes; pois todos deveriam ter internamente as rotinas de controle dos recursos; os programas seriam também complicados e muito extensos.

2.1 FUNCIONALIDADES

O sistema operacional toma conta do computador e facilita a vida do usuário, deixando que ele se preocupe com coisas mais úteis e produtivas que endereços de memória, segmentos e interrupções.

Um sistema operacional (SO) tem que fazer com que o computador (e os periféricos) possa ser usado sem problemas por alguém que não conheça os detalhes do sistema. Ele tem que possibilitar que o usuário crie e acesse arquivos, use programas, acesse a Internet, execute jogos e todas as demais tarefas que podem ser feitas com computadores. Além disso, o SO tem que executar os programas e ajudá-los a acessar os recursos do sistema de uma forma simples e organizada.

O exemplo utilizado para explicar de forma breve, e de fácil entendimento, o funcionamento do sistema operacional feito por Tanenbaum (2000) é apresentado a seguir. O autor refere-se a utilização de uma impressora por três programas, considerando um ambiente que não possui um sistema operacional. Algumas linhas seriam impressas

provenientes da solicitação do programa 1, as próximas do programa 2, então mais algumas do programa 3 e assim por diante. Por isso a necessidade de existir um sistema operacional. Ele mantém a ordem do ambiente computacional, armazena as solicitações feitas para os recursos e as libera de forma que não prejudiquem o funcionamento (ou resultado gerado) dos programas que estão sendo executados.

2.2 PROCESSOS

"Um processo é basicamente um programa em execução."(TANENBAUM, 2000, p. 26). Os computadores podem realizar várias tarefas ao mesmo tempo. Enquanto executa programa de um usuário, pode também estar lendo informações do disco e dar saída de texto para uma impressora sem parar de reproduzir uma música. Na verdade, esta impressão que tem-se sobre o multiprocessamento, deve-se a capacidade da CPU funcionar para vários programas em um curto período de um segundo por exemplo. Porém, em qualquer instante de tempo, apenas um processo está sendo executado. O multiprocessamento existe realmente em ambientes onde 2 ou mais CPU's estejam compartilhando o mesmo espaço de memória (TANENBAUM, 2000).

2.3 HISTÓRIA DOS SISTEMAS OPERACIONAIS

Para usar os primeiros computadores era preciso conhecer profundamente o seu funcionamento, pois a programação era feita em painéis, através de fios. "Naqueles tempos, um único grupo de pessoas projetava, construía, programava, operava e mantinha cada máquina." (TANENBAUM, 2000, p. 20). Após ligar cabos e conectores nos painéis para ajustar o programa, deveriam ainda contar com a sorte para que nenhuma das até 20.000 válvulas queimasse.

Esta forma de programação durou de 1945 a 1950 e ficou conhecida como "A primeira geração", onde a forma de programação teve uma melhora com a introdução dos cartões perfurados. Agora era possível gravar programas em cartões e lê-los, em vez de utilizar cabos e conectores, porém, fora isso, o procedimento era o mesmo.

Nos próximos dez anos (“Segunda geração”), os computadores tiveram uma evolução radical. Chegaram a um ponto onde podiam ser construídos e comercializados, porém apenas grandes corporações tinham recursos suficientes para obtê-los. Neste período também houve a troca da utilização dos cartões perfurados por leituras de fitas magnéticas. (TANENBAUM, 2000).

O período entre 1965 e 1980 foi denominado de “A Terceira Geração”, onde surgiu a multiprogramação: enquanto algum processo aguardava por uma entrada/saída, a unidade de processamento de dados – UCP, podia então estar ocupada realizando outras tarefas.

A quarta geração iniciou-se em 1980 e estende-se até hoje. Nesta fase, surgiram os conhecidos computadores pessoais, de custo acessível e de um bom poder de processamento para serem utilizados em casa. “O desenvolvimento de circuitos LSI (*Large Scale Integration* – integração em larga escala), *chips* contendo milhares de transistores por centímetro quadrado de silício, foi o alvorecer da era do computador pessoal” (TANENBAUM, 2000).

Nesta época surgiram dois sistemas operacionais para microcomputadores: UNIX e MS-DOS. O sucessor do MS-DOS, Windows, inicialmente utilizava-se do MS-DOS para ser executado, mas em 1995 foi lançada a primeira versão independente de modo que o MS-DOS não é mais necessário para suportá-lo. O UNIX é dominante em estações de trabalho e em outros computadores topo de linha, servidores de rede e máquinas com processadores de alto desempenho.

2.4 UNIX

Em 1973 o UNIX foi reescrito em C, com uma quantia relativamente pequena do Kernel – núcleo do sistema operacional, escrita em *assembly* – linguagem de programação de baixo nível, talvez o fato mais importante da história deste sistema operacional. Isto significava que o UNIX poderia ser portado para novo *hardware* em meses, e que mudanças eram fáceis, pois era necessário alterar apenas uma parte do Kernel, em torno de 10%. A linguagem C foi projetada para o sistema operacional UNIX, e portanto, há uma grande sinergia entre C e UNIX.

UNIX é um sistema operacional bom para programadores experientes. O sistema operacional foi projetado e foi implementado por programadores experientes assim todas as necessidades de um programador experiente estão presentes mas, não muito mais. Um exemplo perfeito disto é a documentação *on-line* chamada *man-pages* ou páginas manuais. O material é apenas referência orientada, com muito pouca informação de tutorial. Programadores acham as *man-pages* muito úteis, mas o usuário iniciante as acha muito complicadas (SANTOS, 2004).

2.5 LINUX

A história do Linux teve início na Universidade de Helsinki na Finlândia, onde Linus Torvalds tinha a intenção de desenvolver o Linux como um passatempo. Conforme Danesh (2000, p. 04) “as primeiras versões não se destinavam ao usuário final, mas forneciam o mínimo de funcionalidade para permitir aos programadores de Unix a alegria da programação do kernel.”

O kernel é o núcleo do sistema operacional. Ele mantém tudo funcionando perfeitamente. Danesh (2000, p. 04) diz que “sem um kernel estável e poderoso, você não tem um sistema operacional”.

A tarefa do kernel é fornecer o ambiente global em que os aplicativos possam ser executados, incluindo interfaces básicas com o *hardware* e sistemas de gerenciamento de tarefas e programas em execução.

Em 1992 surgiu a versão 1.0 do kernel, o primeiro lançamento oficial do Linux, que neste ponto executava a maioria das ferramentas Unix, desde compiladores até *softwares* de ligação entre rede e X Windows. O Linux continua evoluindo para computadores pessoais, dando suporte de *hardware*, incluindo periféricos mais populares, dando a muitos computadores pessoais poder comparável à computadores de porte médio.

Danesh (2000) fala que o termo Linux é usado de duas maneiras: especificamente para se referir ao kernel em si e geralmente para se referir a qualquer conjunto de aplicativos que sejam executados no kernel, normalmente chamado de distribuição.

No sentido geral do termo, existem diversas versões do Linux, já que cada distribuição tem características próprias, diferentes conjuntos de recursos, diferentes métodos de instalação e caminhos de atualização.

O Linux é um sistema operacional e possui uma característica de destaque entre outros sistemas operacionais, ele é multitarefa e multiusuário.

Um sistema operacional multitarefa é um sistema que pode executar mais de uma tarefa ou aplicativo por vez. A multitarefa cria a aparência de atividade simultânea, alternando as tarefas conforme a demanda de processos necessária.

O Linux permite vários usuários simultâneos – multiusuário – utilizando os recursos de multitarefa. Sendo assim, ele pode ser distribuído como um servidor de aplicativos, os usuários se conectam a este servidor de seus terminais e através de uma rede local, executam aplicativos no servidor, Danesh (2000).

O Linux pode ser usado para desenvolver praticamente qualquer tipo de aplicativo. Danesh (2000) cita alguns aplicativos disponíveis:

A. aplicativos de processamento de texto: além de *software*s de processamento de texto comercial como *WordPerfect*, *StarOffice* e *Applixware*, o Linux oferece ferramentas para edição e processamento de arquivos de texto;
B. linguagens de programação: ampla variedade de linguagens de programação e de *script* e ferramentas disponíveis para Linux e todos os sistemas operacionais Unix.
C. *X Windows*: o *X Windows* é um ambiente gráfico altamente flexível e configurável que é executado no Linux, assim como a maioria dos sistemas Unix.
D. ferramentas para Internet: fornece suporte a *software*s como o Netscape Communicator e Mosaic e, além disso, possui uma ampla gama de *software* para internet, desde aplicativos para leitura de correios até *software*s para a criação de servidores de internet e suporte de rede para conexão à internet através de rede local ou modem.

E. banco de dados: bancos de dados como o mSQL, Postgre, Oracle, Sybase e Informix estão disponíveis para Linux.
F. *software* de compatibilidade com DOS e Windows: o Linux pode executar *software* DOS com alto grau de estabilidade e compatibilidade e oferece várias estratégias para a execução de *software* Microsoft Windows.

Ao contrário de outros sistemas operacionais, o kernel do Linux e a maioria dos aplicativos estão disponíveis gratuitamente na Internet, normalmente sem restrições sobre cópias do *software*.

O kernel do Linux é distribuído sob a GNU - *General Public License*, esta licença desenvolvida pela *Free Software Foundation* promove a distribuição e desenvolvimento aberto do *software*. A GNU permite que qualquer um redistribua um *software*, isto significa que qualquer um pode pegar um *software* GNU, alterá-lo e distribuí-lo, mas não pode impedir que outra pessoa adquira seu *software* licenciado pela GNU e faça o mesmo Danesh (2000).

2.6 ESCALONADORES

Os escalonadores de processos são responsáveis por determinar qual processo será o próximo a ser iniciado. Alguns escalonadores visam um melhor aproveitamento de recursos computacionais e outros se preocupam com a prioridade dos processos.

Conforme Deitel (1990, p.287), existem três níveis importantes de escalonadores que serão descritos abaixo:

A) *High level scheduling* (escalonamento de alto nível) – também são conhecidos por escalonadores de admissão porque eles determinam qual processo ganha admissão no sistema operacional. Estes escalonadores determinam quais processos poderão competir pelos recursos disponíveis;

B) *Intermediate-level scheduling* (escalonamento de nível intermediário) – este tipo de escalonador determina quais processos poderão competir pelo uso da CPU. É ele quem ativa os processos, suspende-os quando necessário e os

reinicia. Ele atua como um *buffer* entre a admissão dos processos no sistema e a associação da CPU com estes processos;

C) *Low-level scheduling* (escalonamento de baixo nível) – Determina qual processo pronto será associado a CPU quando estiver disponível.

3 ADMINISTRAÇÃO DE AMBIENTES COMPUTACIONAIS

Ables (1995), cita que muitos administradores de sistema acham difícil definir e quantificar as tarefas que possuem. Alguns administradores acreditam até que não exista um conceito fácil para resumir quais são suas tarefas. “Deixar os usuários felizes” pode até parecer um termo estranho, mas no final de tudo, isto é exatamente o que o administrador de sistemas deve fazer.

Há quem acredite que administração de sistemas possa se resumir da seguinte forma: se os usuários estão felizes com seu trabalho e isto os tem ajudado a cumprir as tarefas deles, você fez um bom trabalho. É claro que existem usuários que estão tentando encontrar a resposta para seus problemas em “problemas com o ambiente” e estes não estarão felizes o tempo todo. Existem também os usuários que estão literalmente querendo ajudar a melhorar algum tipo de processo; mas na verdade, todos eles originam uma boa demanda de trabalho para os administradores. Nesta demanda de trabalho intermitente, cabe ao administrador de sistemas saber classificar os problemas recebidos em prioridades, Ables (1995).

As tarefas de um administrador de sistema não precisam ser realizadas sempre pela mesma pessoa ou apenas por ela. Esta “dependência” pela pessoa do “administrador de sistema” não é confortável para empresa e nem para o administrador; mas muitas vezes, isto ainda ocorre. Entretanto, em alguns ambientes computacionais, a demanda de trabalho de administração faz com que se empregue mais de um recurso para poder realizar todas as tarefas, Nemeth (2002, p.50).

3.1 TAREFAS ESSENCIAIS DO ADMINISTRADOR DE SISTEMA

Nemeth (2002) descreve algumas das principais tarefas de um administrador de sistema:

a) adicionar e remover usuários: o administrador deve adicionar contas para novos usuários e remover as que não estão mais sendo utilizadas;

b) remover e adicionar *hardware*: na aquisição de um novo *hardware* ou na transferência de uma máquina para outra, o sistema deve ser configurado para atender a esta alteração;

c) realizar *backups*: este talvez possa ser considerado o trabalho mais importante do administrador de sistemas. Os *backups* são demorados e muitas vezes complicados mas precisam ser feitos pois são muito importantes;

d) instalar novo *software*: na aquisição de um novo *software*, ele deve ser instalado e testado antes de estar liberado para que os usuários o utilizem. Normalmente o *software* utilitário é instalado em um lugar que o torne fácil de ser diferenciado do *software* de sistema;

e) monitoração do sistema: atividades diárias incluem certificar-se de que os serviços de correio eletrônico e de *Web* estão trabalhando corretamente, observar arquivos de *log* para detectar sinais recentes de problemas, assegurar que redes locais estão todas adequadamente conectadas e ficar de olho na disponibilidade de recursos do sistema, como o espaço em disco;

f) solucionar problemas: é tarefa do administrador identificar problemas e chamar especialistas se necessário. Localizar o problema normalmente é mais simples do que soluciona-lo;

g) manter documentação local: todas alterações feitas no ambiente devem ser documentadas pelo administrador. À medida que o sistema for alterado para atender às necessidades da organização, ele começará a diferir do sistema básico pela documentação;

h) fazer auditoria de segurança: o administrador deve implementar uma política de segurança e periodicamente verifica-la para certificar-se de que a segurança do sistema não tenha sido violada;

i) ajudar os usuários: embora raramente esta tarefa seja incluída em uma descrição de trabalhos para um administrador de sistema, esta é uma tarefa que consome uma parte significativa dos dias úteis dos administradores.

Como visto, a administração de sistemas exige uma boa flexibilidade do administrador para contornar os problemas e as tarefas do ambiente. Porém, a administração de ambientes computacionais pode ser simplificada com o uso de *scripts* para auxiliar nas tarefas executadas com uma maior freqüência no ambiente, e também com uma correta avaliação dos arquivos de *logs* gerados pelas aplicações.

3.2 SCRIPTS

Segundo Danesh (2000, p. 169), um item administrativo fundamental em um ambiente é a automação de tarefas. Isto é necessário pois as tarefas de administração de um ambiente são inúmeras e muitas delas são repetidas várias vezes da mesma forma.

Para isto, é recomendado que os procedimentos repetidos com maior freqüência, sejam escritos de uma forma que possam ser executados mais facilmente a cada vez que se tornem necessários. Um dos pilares da administração de sistemas do tipo UNIX, é o uso de *scripts* para automatizar tarefas administrativas, Nemeth (2002, p.44). Algumas das linguagens mais utilizadas para o desenvolvimento destes scripts são: shell unix, c e perl.

3.3 ARQUIVOS DE *LOG*

Vários utilitários que estão instalados em um ambiente, assim como o próprio sistema operacional, emitem informações sobre suas execuções que são gravadas e por fim armazenadas em discos com tamanho finito. Estas informações são conhecidas como arquivos de *logs*. Estes arquivos precisam ser avaliados, resumidos, compactados e descartados freqüentemente pela pessoa do administrador de sistema.

Ambientes sujeitos a problemas de segurança rotineiramente descobrem que dados dos arquivos de *log* fornecem evidências importantes sobre invasões. Os arquivos de *log* também são úteis para alertar sobre problemas de *hardware* e *software*, Nemeth (2002).

O LINUX fornece sofisticados recursos de *log* que tornam possível saber exatamente o que está ocorrendo no sistema, porém, para que estes *logs* sejam úteis, precisam de um gerenciamento regular, Danesh (2000, p. 169).

4 DESENVOLVIMENTO DO TRABALHO

Neste capítulo são apresentados os requisitos, a especificação, tecnologias utilizadas, implementação, operacionalidade do sistema e funcionalidades da ferramenta criada, resultado e discussões.

4.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

A ferramenta criada tem como objetivo auxiliar os administradores de ambientes computacionais fazendo algumas tarefas que são precedidas por algum evento conhecido.

O administrador do sistema deverá configurar o escalonador para indicar qual o intervalo de tempo em segundos entre os ciclos de monitoração.

É de responsabilidade do administrador do ambiente cadastrar os processos e informar onde ficarão os arquivos com as informações das ações tomadas pela ferramenta.

O administrador também deverá efetuar o cadastro dos sub-processos informando os parâmetros de cada um deles, obedecendo a ordem dos mesmos, informando também o caminho do *log* de cada um deles.

Todos os programas a serem disparados pelo escalonador deverão ser do *owner* (dono do arquivo) *root* por motivos de segurança.

Informações sobre as execuções dos processos poderão ser feitas através de *logs* gerados ou pelas telas do sistema.

A ferramenta deverá ler as tabelas de configuração, de processos e sub-processos a cada ciclo e agregar a sua lista de processos os novos processos cadastrados, assim como se adaptar às novas configurações sem precisar ser reiniciado.

Foram identificadas quatro situações freqüentes que podem ser monitoradas por uma ferramenta automática. Estas situações são apresentadas a seguir com uma possível ação associada:

a) situação 1: o sistema gerou uma mensagem de erro ou de advertência. É possível relacionar as possíveis saídas que representarão condições não desejadas e quando identificadas, executar uma ação para corrigir ou recuperar. Para tanto, as saídas geradas pelo programa em execução a ser monitorado, devem ser previamente enviadas para um arquivo denominado *log*. Um processo ficará periodicamente examinando o conteúdo deste arquivo para identificar a existência de uma mensagem pré-cadastrada. Caso encontre será executado um programa previamente cadastrado, que está associado a esta mensagem;

b) situação 2: uma aplicação qualquer parou de atualizar o arquivo de *log*. Pode-se relacionar o nome de um arquivo de *log* a um tempo em que este arquivo deve, impreterivelmente, ser atualizado. Caso seja verificado que o arquivo não foi atualizado no tempo especificado, isto indica que a aplicação está com problemas. Então, a ação associada ao nome deste arquivo de *log* será disparada para corrigir o problema;

c) situação 3: um processo qualquer parou de funcionar. É possível informar o nome do processo que deve ser monitorado e a ação que deve ser executada assim que for verificado que este processo encontra-se inativo. Freqüentemente este processo estará sendo monitorado e caso fique inativo, a ação associada à ele será disparada;

d) situação 4: a utilização de uma estrutura física do ambiente atingiu seu limite da capacidade de armazenamento. Para este tipo de problemas, pode-se informar o nome da estrutura física e o percentual crítico de sua utilização. Periodicamente esta estrutura será monitorada e caso for identificado que o percentual de sua utilização seja igual ou maior do que o associado à ela, o evento associado a esta estrutura será executado para corrigir o problema.

Para as ações cadastradas que não estiverem associadas a alguma das situações listadas acima, a ferramenta irá apenas disparar o evento cadastrado com seus parâmetros.

Com a proposta ferramenta administrativa é *freeware* (gratuita) e *opensource* (de código aberto, que pode ser alterado livremente), existirá ainda a possibilidade de serem

cadastrados vários tipos de processos para agruparem sub-processos. Caberá ao administrador apenas, alterar o código do escalonador para que ele passe a tratar o novo tipo de ação a ser tomada para os novos eventos cadastrados.

4.2 ESPECIFICAÇÃO

A ferramenta Power Designer foi utilizada para a elaboração dos diagramas: entidade-relacionamento, lógico e físico; diagrama de fluxo de dados completo e particionado. A lista de eventos foi montada com base na metodologia da análise estruturada.

4.2.1 LISTA DE EVENTOS

Na lista de eventos descrita abaixo, são identificados os acontecimentos desde a configuração inicial do sistema:

a) Administrador configura escalonador;
b) Administrador cadastra processos;
c) Administrador cadastra sub-processos;
d) Escalonador monitora sub-processo;
e) Escalonador dispara evento associado ao sub-processo;
f) Escalonador atualiza a base de dados.

No primeiro evento, o administrador configura o escalonador conforme as necessidades do ambiente.

Nos próximos dois eventos, são cadastrados os processos, que definem um grupo de problemas. No cadastro de sub-processos, é detalhada uma situação para o evento “pai” processo, onde será informada também a solução que deve ser tomada para este sub-processo, caso a ação “processo” ocorra para este sub-processo.

No quarto evento, o escalonador verifica a condição de cada sub-processo cadastrado e a avalia no ambiente computacional. Se for identificada a necessidade de atuar sobre este sub-processo, ocorre então o quinto evento: disparar a ação associada a este sub-processo.

O sexto evento é a atualização da base por parte do escalonador, com as informações da ação que teve que tomar para cada sub-processo.

As características esperadas para esta ferramenta são:

a) facilidade na utilização;
b) confiabilidade, não disparar processos indevidamente;
c) fornecimento de informações sobre os sub processos submetidos;
d) ações tomadas rapidamente ao identificar um problema.

4.2.2 DFD PARTICIONADO

Um exemplo de DFD particionado, mostrando o do cadastramento de sub-processos é apresentado abaixo na figura 1.

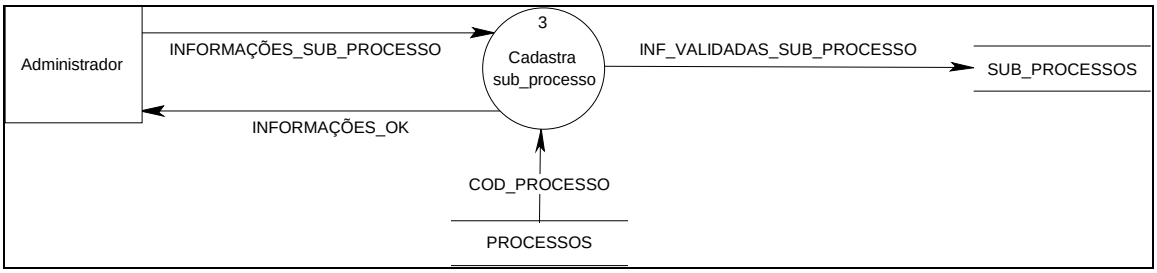

Figura 1 - DFD particionado

4.2.3 DFD DE NÍVEL

O DFD completo do sistema é visto na figura 2.

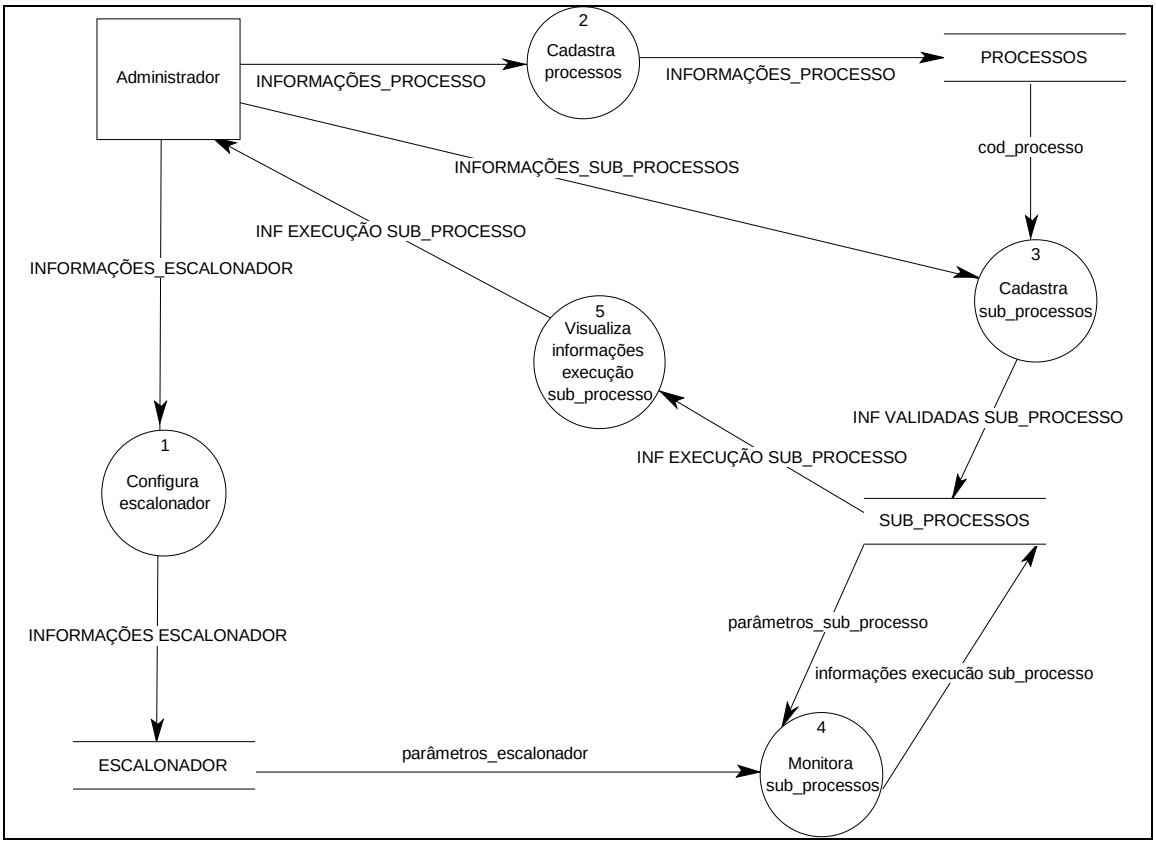

Figura 2 - DFD de nível

4.2.4 DER LÓGICO

A figura 3 apresenta a estrutura e relacionamento das entidades.

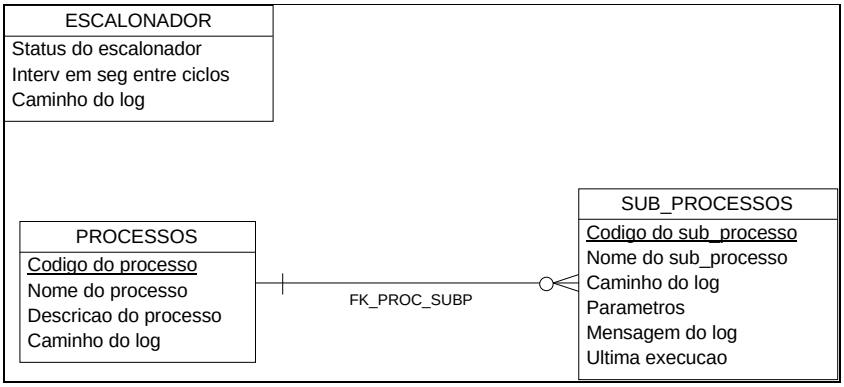

Figura 3 - DER lógico

4.2.5 DER FÍSICO

A tradução do modelo lógico para o físico, utilizando o banco de dados PostgreSQL é mostrada na figura 4.

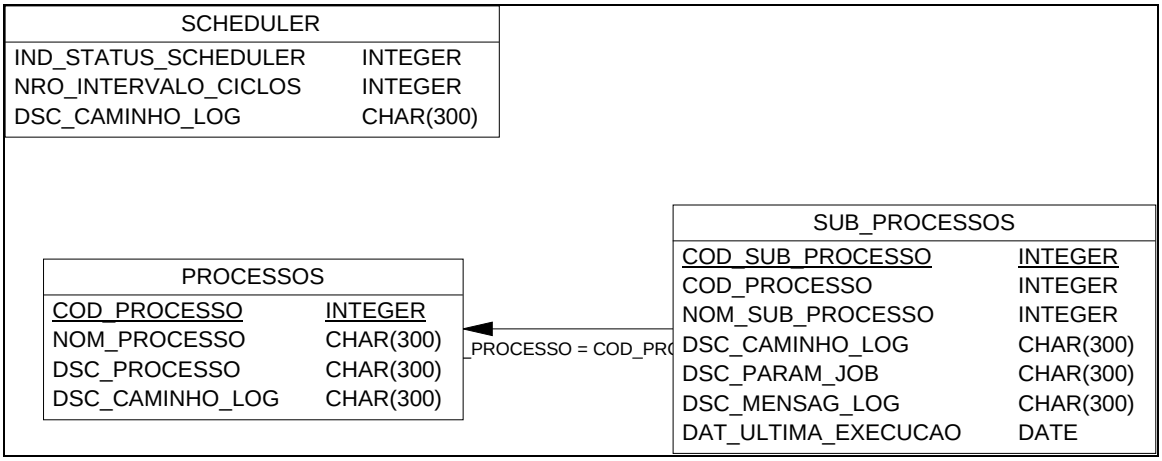

Figura 4 - DER Físico

4.2.6 DICIONÁRIO DE DADOS

Abaixo têm-se o dicionário de dados da estrutura total do sistema.

Tabela Escalonador

Column List

Name	Code	Type	P	M
Status do escalonador	IND_STATUS_SCHEDULER	INTEGER	No	Yes
Interv em seg entre ciclos	NRO_INTERVALO_CICLOS	INTEGER	No	Yes
Caminho do log	DSC_CAMINHO_LOG	CHAR(300)	No	Yes

Tabela Processos

Column List

Name	Code	Type	P	M
Código do processo	COD_PROCESSO	INTEGER	Yes	Yes
Nome do processo	NOM_PROCESSO	CHAR(300)	No	Yes
Descricao do processo	DSC_PROCESSO	CHAR(300)	No	Yes
Caminho do log	DSC_CAMINHO_LOG	CHAR(300)	No	Yes

Tabela Sub-processos

Column List

Name	Code	Type	P	M
Código do sub_processo	COD_SUB_PROCESSO	INTEGER	Yes	Yes
Código do processo	COD_PROCESSO	INTEGER	No	Yes
Nome do sub_processo	NOM_SUB_PROCESSO	INTEGER	No	Yes
Caminho do log	DSC_CAMINHO_LOG	CHAR(300)	No	Yes
Parâmetros	DSC_PARAM_JOB	CHAR(300)	No	Yes
Mensagem do log	DSC_MENSAG_LOG	CHAR(300)	No	No
Ultima execução	DAT_ULTIMA_EXECUCAO	DATE	No	No

4.3 IMPLEMENTAÇÃO

O fluxograma representado nas figuras 5, 6 e 7 mostra o fluxo de informações do sistema. No fluxograma observa-se que o procedimento inicia na leitura das configurações do escalonador. Se o indicador for igual a 2 o escalonador é finalizado; se estiver em 0 (zero), aguarda o número de segundos lidos na tabela do escalonador e repete a operação; se estiver em 1, inicia as verificações dos sub-processos.

Se não existirem sub-processos cadastrados, o escalonador volta a aguardar o tempo de espera para ler novamente as configurações do escalonador; se existirem sub-processos cadastrados, a verificação principal do sistema inicia.

Se o código do sub-processo for igual a 1, o parâmetro “arquivo” recebe o parâmetro1, “expressão” recebe o parâmetro2 e “ação” recebe o parâmetro3. É procurada então a “expressão” no “arquivo”. Caso seja encontrada, a “ação” é executada, o banco é atualizado com as informações deste sub-processo e um próximo sub-processo é lido. Se a “expressão” não for encontrada no “arquivo”, o próximo sub-processo é lido.

Se o código do sub-processo for igual a 2, o parâmetro “arquivo” recebe o parâmetro1, “tempo” recebe o parâmetro2 e “ação” recebe o parâmetro3. É verificada a data da última atualização do “arquivo” no sistema operacional. Caso o intervalo de tempo entre a última atualização e a data atual seja maior ou igual ao “tempo”, a “ação” é executada, o banco é atualizado com as informações deste sub-processo e um próximo sub-processo é lido. Se o intervalo de tempo não for excedido, o próximo sub-processo é lido.

Se o código do sub-processo for igual a 3, o parâmetro “programa” recebe o parâmetro 1 e o parâmetro “ação” recebe o parâmetro2. É executada uma chamada no S.O para verificar se este “programa” está ativo. Caso o “programa” não esteja ativo, a “ação” é executada, o banco é atualizado com as informações deste sub-processo e um próximo sub-processo é lido. Se o “programa” estiver ativo, o próximo sub-processo é lido.

Se o código do sub-processo for igual a 4, o parâmetro “estrutura” recebe o parâmetro1, “% crítico” recebe o parâmetro2 e “ação” recebe o parâmetro3. É verificado no sistema operacional o percentual de utilização da “estrutura”. Caso o percentual de utilização desta estrutura seja igual ou maior do que o “% crítico”, a “ação” é executada, o banco é atualizado com as informações deste sub-processo e um próximo sub-processo é lido. Se o percentual de utilização não estiver igual ou maior do que o “% crítico”, o próximo sub-processo é lido.

Para qualquer outro código não cadastrado o sistema associa o parâmetro1 com “programa” e o parâmetro2 como “parâmetros”. Será então feita uma chamada no sistema operacional para disparar este “programa” com seus “parâmetros. O banco de dados é atualizado com as informações desta submissão e um próximo sub-processo é procurado.

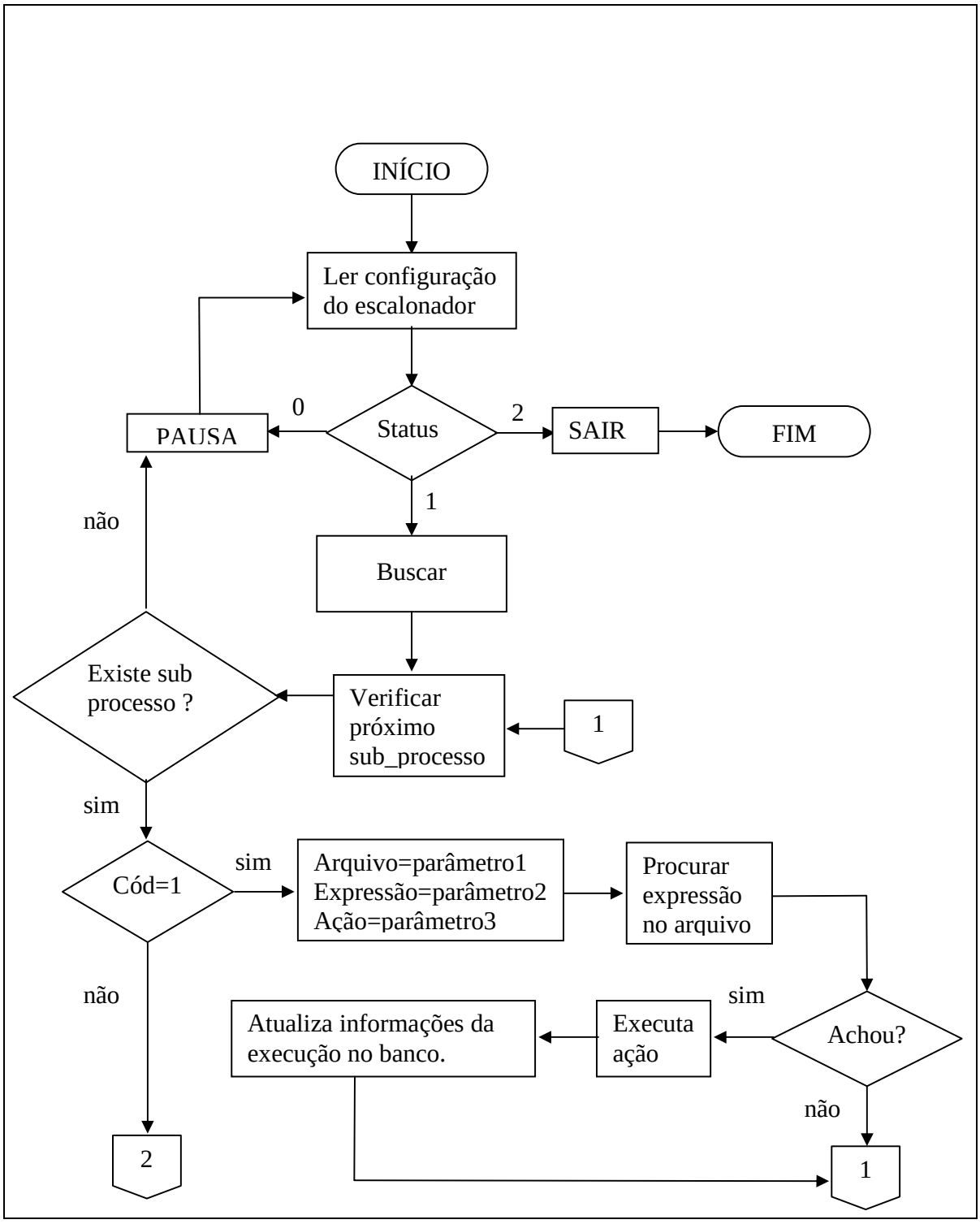

Figura 5 - Fluxograma

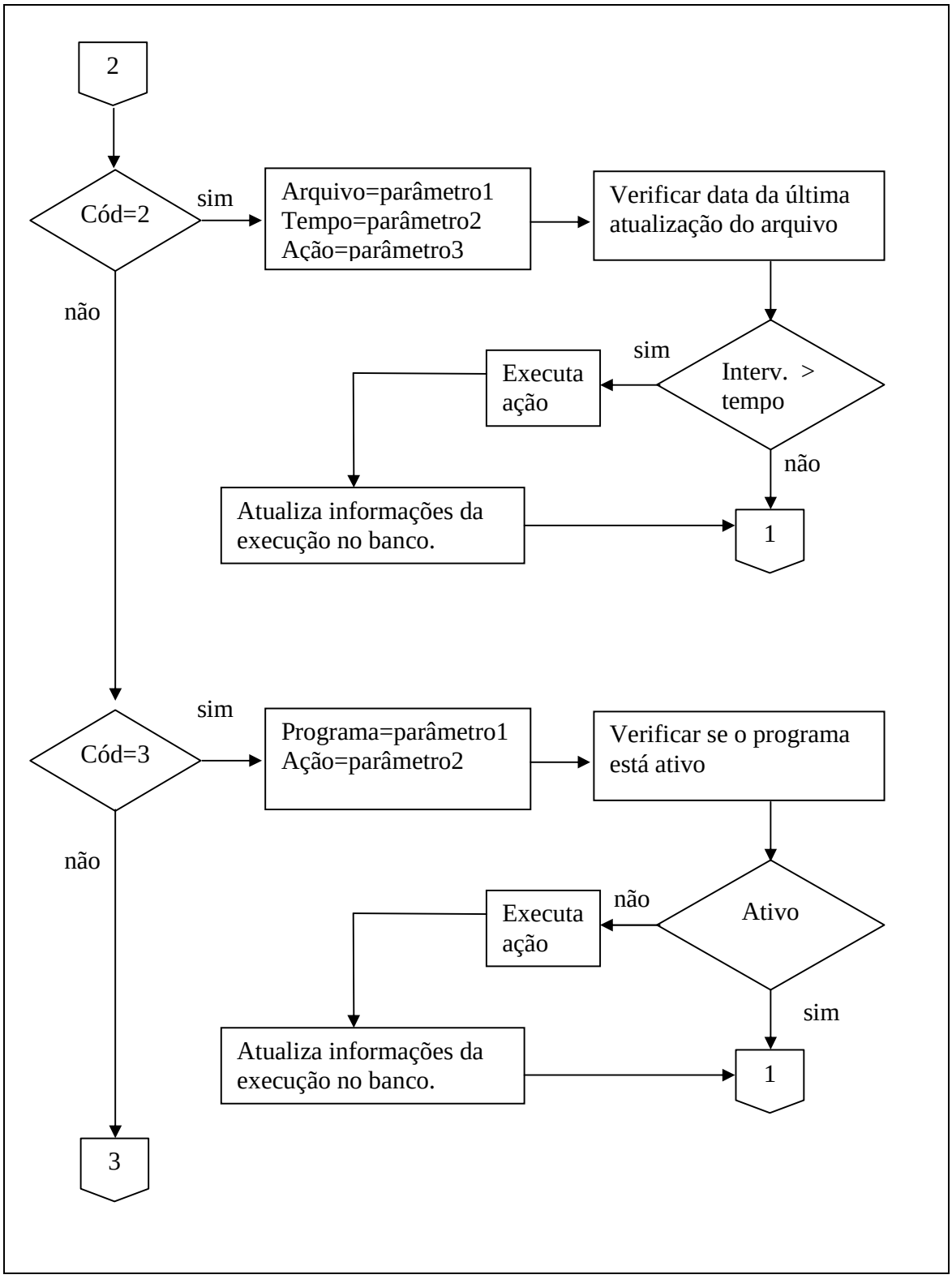

Figura 6 - Fluxograma (continuação)

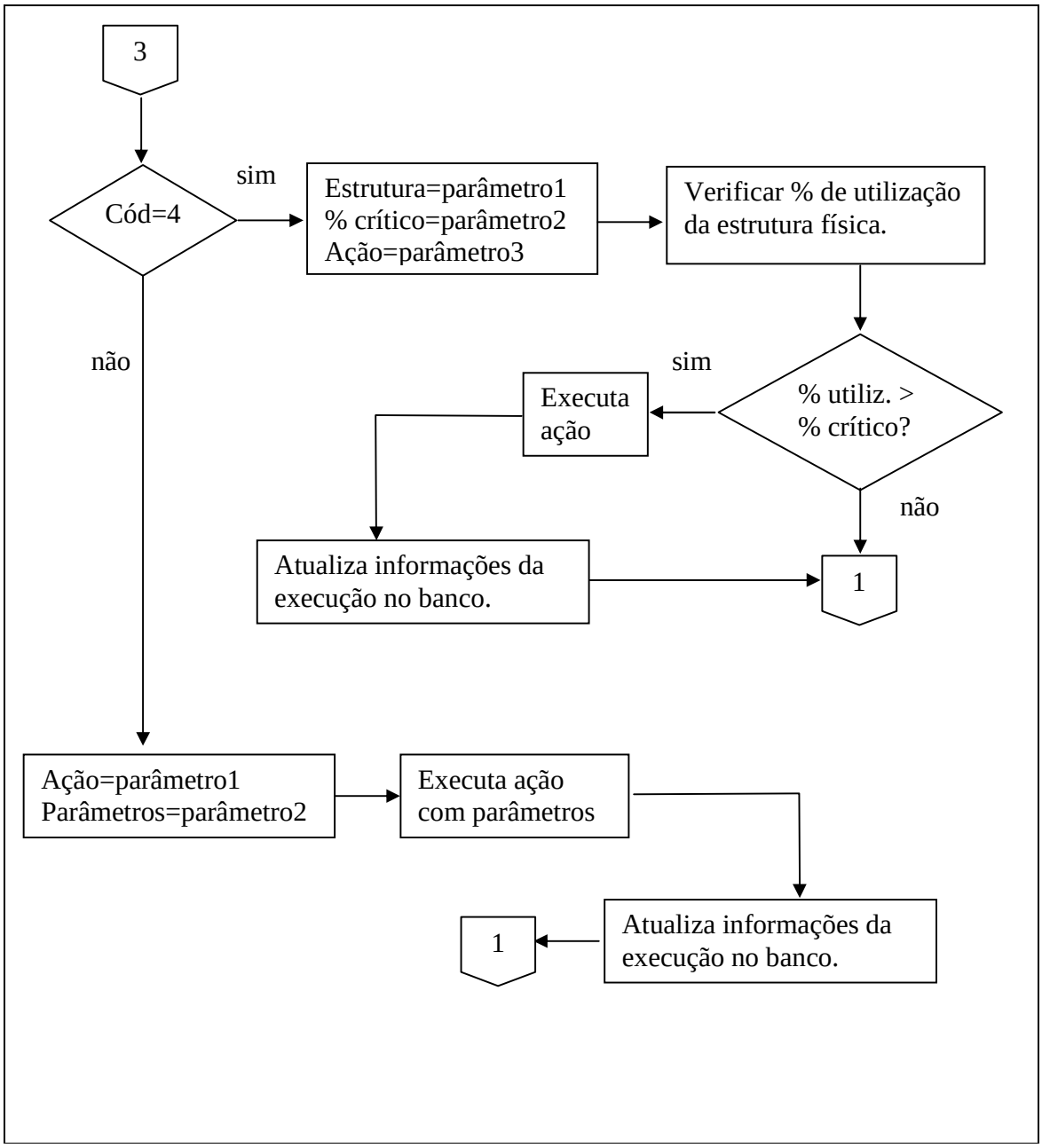

Figura 7 - Fluxograma (continuação)

O trecho de código exibido abaixo na figura 8 mostra como é tratada a situação de procura por uma expressão dentro de um arquivo:

```
case 1: printf("%d-%s \n",p_cod_processo, p_dsc_processo);
        //  wrk_a=arq de log, wrk_b=string a ser procurada, wrk_c=cmd a ser ex, se a string for encontrada.
        strcpy(wrk_comando, "grep -q "); // montar comando para executar no s.o.
        strcat(wrk_comando, wrk_b);
        strcat(wrk_comando, " ");
        strcat(wrk_comando, wrk_a);
        execucao = system(wrk_comando);  // executar comando e guardar retorno em execucao
        if (execucao  == 0 )             // se a string foi encontrada, executa processo contido em wrk_c
          {
            strftime(string_temp, 20, "%m/%d/%y %h:%m:%s", dat);  // formata data recebida
            strcat(string_temp," encontrou string [");
            strcat(string_temp, wrk_b);
            strcat(string_temp, "] no log [");
            strcat(string_temp, wrk_a);
            strcat(string_temp, "]\n");
            printf(string_temp);
            if ((fp = fopen(arq_log_subprocesso,"a"))==null){
              printf("erro abrindo arquivo %s\n",arq_log_subprocesso);
            }
            fprintf(fp,string_temp); // atualiza arquio de log com informacoes
            fprintf(fp,"\n");
            fclose(fp);
            strcpy(wrk_comando, "cp  ");
            strcat(wrk_comando, wrk_a );       // copiar arquivo de log antigo
            strcat(wrk_comando, " ");          // para log antigo.timestamp
            strcat(wrk_comando, wrk_a );       // no mesmo diretorio do log original
            strcat(wrk_comando, ".");          // para que o processo nao leia novamente
            strcat(wrk_comando, timestamp);    // a string ja encontrada.
            system(wrk_comando);
            fp = fopen(wrk_a, "w");            // zerar arquivo de log
            fclose(fp);                        // fecha arquivo para liberar mem.
            exec sql update sub_processos
            set    dat_ultima_execucao=now(), dsc_mensag_log='string encontrada!'
            where  cod_processo=:p_cod_processo
            and    cod_sub_processo=:p_sub_cod_sub_processo;
            strcpy(wrk_comando, wrk_c);        // copia comando para wrk_comando
            strcat(wrk_comando, " &");
            system(wrk_comando);               // executa procedimento def. pelo usuario.
          }
```

Figura 8 - Trecho de código do sistema

4.4 TÉCNICAS E FERRAMENTAS UTILIZADAS

Para a implementação do sistema foi utilizado o banco de dados PostgreSQL e as linguagens de programação C, PHP e HTML. Para a compilação do código C, foi utilizado o compilador gcc do linux.

Análise estruturada foi a metodologia utilizada para a especificação do sistema. Esta especificação foi feita na ferramenta *Power Designer*.

O sistema operacional utilizado foi o LINUX, pois ambientes de médio e grande porte normalmente utilizam este tipo de sistema. O banco de dados escolhido foi o PostgreSQL por sua confiabilidade, praticidade e por ser um *software* livre. Também

contou-se com as linguagens HTML e PHP para a produção das telas de configuração do sistema; e o desenvolvimento do escalonador foi feito com a linguagem C .

4.4.1 C

Conforme Schildt (1996), C permite a manipulação dos elementos básicos com os quais o computador funciona, por isso um código em C é muito portável. Portabilidade significa que é possível adaptar um programa de computador escrito para um tipo de computador a outro.

Schildt (1996) afirma ainda que um outro aspecto importante da linguagem C, é que possui apenas 32 palavras-chaves (comandos que compõem a linguagem). E isto a torna de mais fácil utilização do que a maioria das linguagens de alto nível, que chegam a possuir mais de 100 palavras-chaves.

4.4.2 HTML

HTML – *Hypertext Markup Language* - é a linguagem da *World Wide Web* por meio da qual podemos desenvolver páginas e exibi-las com perfeição na maioria dos *browsers* (aplicação utilizada para visualizar/acessar páginas) disponíveis no mercado, (SILVA, 2001).

Segundo Ramalho (2001), a linguagem é bastante simples e tem como finalidade básica formatar o texto exibido e criar ligações entre as páginas *WEB*.

Segundo Silva (2001, p.7), o HTML nunca permitirá construir aplicações que rodam sozinhas, que sejam executáveis. Para que se faça isso, é necessário utilizar alguma outra linguagem como por exemplo Java Script ou PHP.

4.4.3 PHP

PHP (PHP *Hypertext Preprocessor*) é uma linguagem de *script* para servidor que facilita a criação de páginas Web dinâmicas embutindo códigos PHP em documentos HTML.

> "A linguagem PHP foi concebida durante o outono de 1994 por Rasmus Lerdorf. As primeiras versões não foram disponibilizadas, tendo sido utilizadas em sua *home-page* apenas para que ele pudesse ter informações sobre as visitas que estavam sendo feitas. A primeira versão utilizada por outras pessoas foi disponibilizada em 1995, e ficou conhecida como *Personal Home Page Tools* (ferramentas para página pessoal). Era composta por um sistema bastante simples que interpretava algumas macros e alguns utilitários que rodavam por trás das *home-pages*: um livro de visitas, um contador e algumas outras coisas" (MEDEIROS, 2004).

Segundo Medeiros (2004), PHP é uma linguagem que permite criar sites WEB dinâmicos, possibilitando uma interação com o usuário através de formulários, parâmetros da URL e links. A diferença de PHP com relação a linguagens semelhantes a Javascript é que o código PHP é executado no servidor, sendo enviado para o cliente apenas html puro. Desta maneira é possível interagir com bancos de dados e aplicações existentes no servidor, com a vantagem de não expor o código fonte para o cliente. Isso pode ser útil quando o programa está lidando com senhas ou qualquer tipo de informação confidencial.

4.4.4 POSTGRESQL

O PostgreSQL é um *Open Source* – ferramenta de código aberto, e implementa os padrões SQL ANSI 92,96 e 99. O SQL (*Structured Query Language*) que é a linguagem estruturada padrão para acessar os dados em um sistema gerenciador de banco de dados relacional que foi criado no início dos anos 80, pela IBM. Sua instalação em Unix e Linux é bastante simples, o que já não ocorre na plataforma MS-Windows, onde são necessários pacotes adicionais.

> O postgreSQL é um banco de dados relacional que evoluiu de forma espantosa nos últimos anos, incorporando características somente disponíveis em bancos de dados com mais tradição e extremamente caros. (SOARES, 2002, p. 107)

O que torna o postgreSQL atraente atualmente é que além de seguro e versátil ele é gratuito e de fácil instalação.

Conforme Neves (2002) é importante definir alguns conceitos para o entendimento do PostgreSQL :

A) sistema Gerenciador de Banco de Dados (SGDB): permite que bancos de dados “persistentes” sejam concorrentemente partilhados por muitos usuários e aplicações utilizando manuseio de armazenamento e estratégias de otimização;
B) banco de dados relacional: é sustentado pelo modelo relacional, matematicamente perfeito, completo e consistente internamente;
C) banco de dados orientado a objetos (OODBs) : integra a orientação a objeto com aptidões de bando de dados, oferecendo modelos diretos e intuitivos para o desenvolvimento de aplicações.

Neves (2002, p.18) diz que “é um dos melhores bancos de dados para sistema operacional GNULINUX, voltado para aplicações simples ou complexas com uma robustez excepcional, suporte a várias linguagens de programação como C, Tcl , SQL, etc. O PostgreSQL permite herança de tabelas e um banco de dados relacional-objeto”.

Em 1994 um interpretador de SQL foi adicionado ao Postgres e em 1995 o desenvolvimento do Postgres foi iniciado.

Em 1996 o Postgres95 foi substituído pelo PostgreSQL. Pereira Neto (2003, p. 28) cita que “tratava-se do código-fonte original do Postgres construído pela Universidade de Berkeley, acrescido do código fonte do interpretador SQL.”

Para Neves (2002), uma sessão de Postgres consiste em :

a. um processo *daemon* – aplicação de supervisionamento (*postmaster*), a aplicação sobre a qual trabalha o utilizador e um ou mais servidores de base de dados de *backend*- processo que fica ativo mas não aparentemente visível, não possui uma interface gráfica;
b. um único *postmaster* administra uma determinada coleção de base e dados em um único servidor. As aplicações *frontend* – aplicações utilizadas para interagirem com usuários externos, que desejam ter acesso a uma determinada base de dados fazem ligações para uma biblioteca, esta envia pedidos através de

uma rede, o qual em resposta começa um novo processo no servidor (*backend*) e conecta o processo de *frontend* ao novo servidor;

c. a partir daqui o processo *frontend* e o servidor de *backend* se comunicam sem a intervenção do *postmaster*.

4.5 OPERACIONALIDADE DA IMPLEMENTAÇÃO

Na tela de apresentação do sistema, figura 9, pode-se escolher as opções de configuração e acompanhamento do escalonador. A seqüência de telas a ser mostrada é: tela de apresentação, configuração do escalonador, configuração de processos, configuração/acompanhamento de sub-processos.

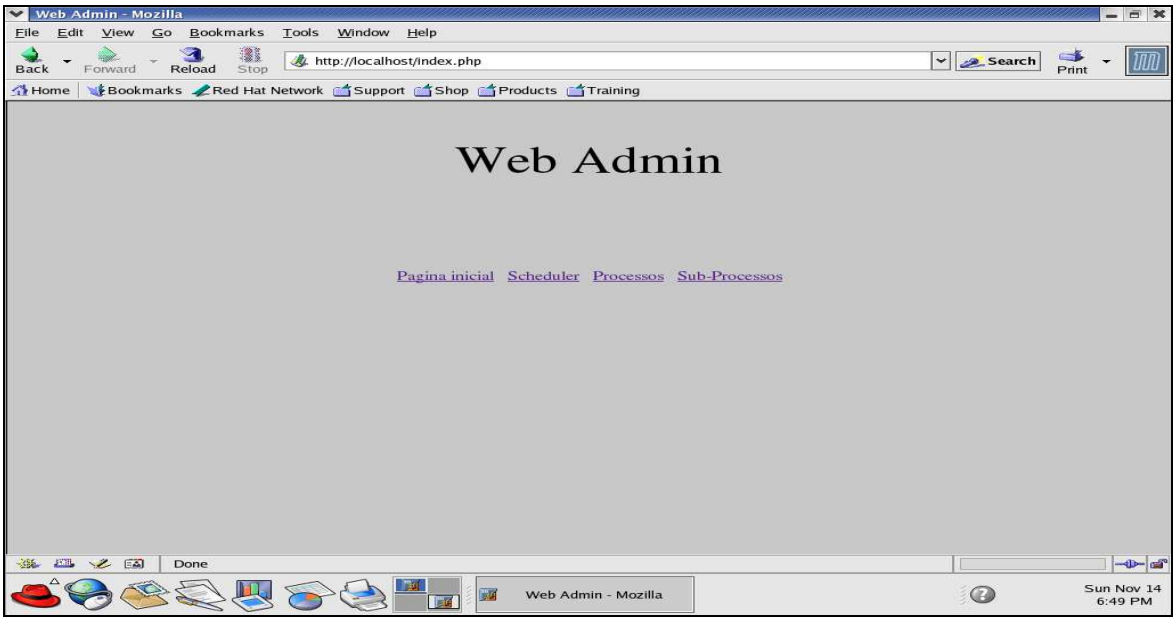

Figura 9 - Tela principal da Ferramenta administrativa

Na tela de configuração do escalonador, figura 10, deve-se informar o tempo (em segundos) que o escalonador deverá esperar entre um ciclo e outro e também o estado atual do escalonador: 0 - para inativo, 1 - para ativo e 2 - para o encerrar.

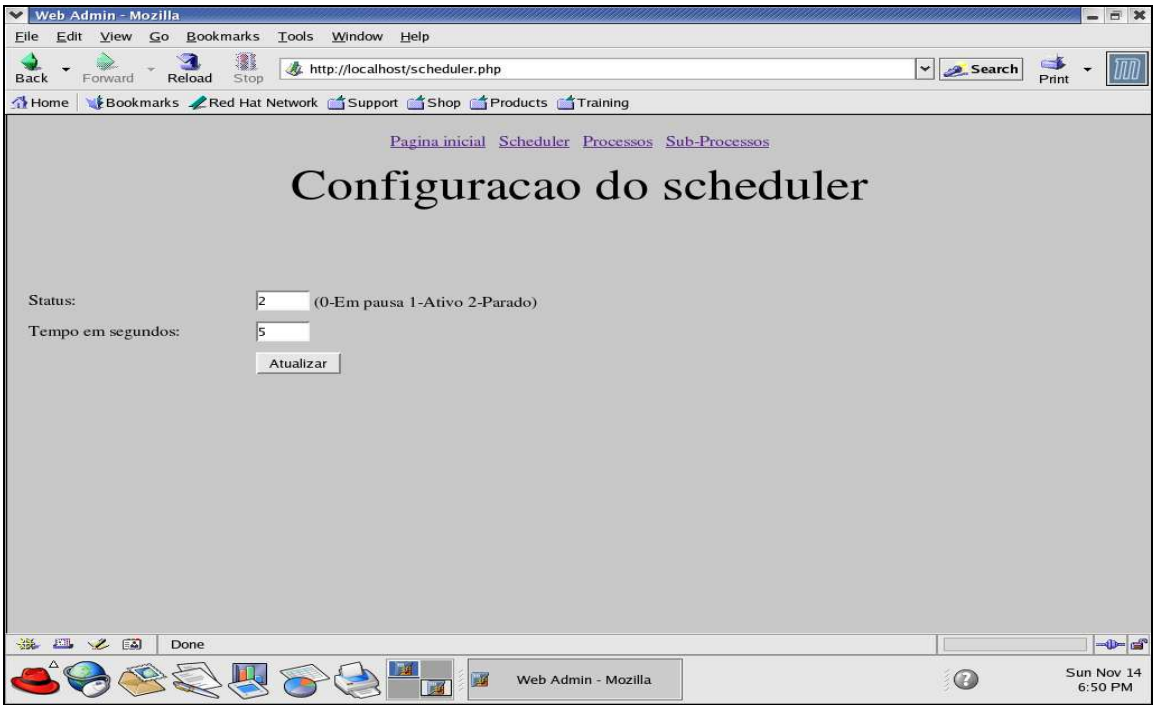

Figura 10 - Tela de configuração do escalonador

A tela de cadastros de processos, figura 11, já estará com alguns processos pré-definidos (aqueles que foram disponibilizados pela ferramenta) que estão listados abaixo:

a) buscar por uma expressão em um arquivo de log;

b) verificar data de última alteração de um arquivo de log;

c) iniciar programa se ele estiver inativo;

d) verificar espaço em disco.

Para processos novos, deverá ser informado um código para o tipo de processo, o nome deste processo que deve ser o nome do programa (com o caminho da estrutura do sistema operacional onde ele encontra-se fisicamente: ex: /usr/local/bin/processo_novo.sh) que será disparado nesta situação, a descrição do tipo do processo e o caminho do *log* principal deste processo.

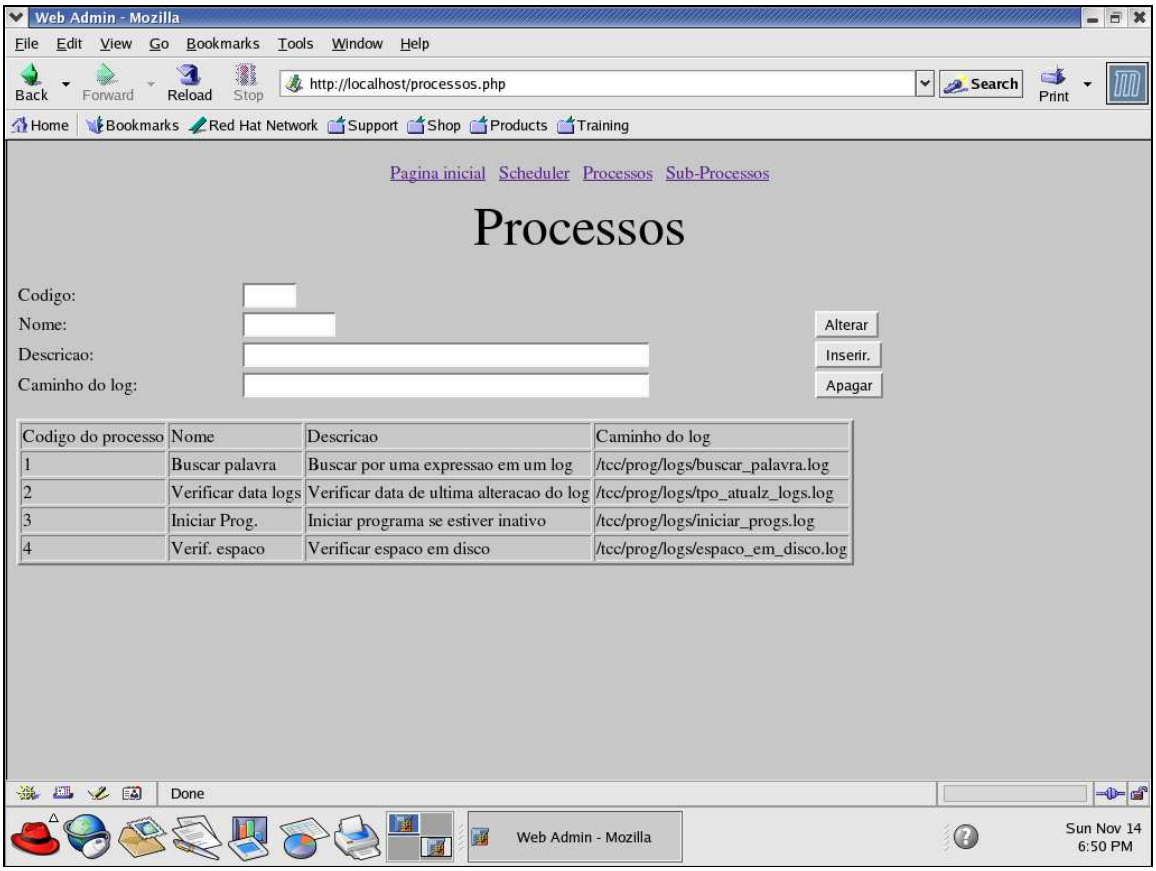

Figura 11 - Tela de configuração dos processos

Na figura 12, mostrada a seguir, é apresentada a tela de cadastro dos sub-processos. Deverá ser informado o código do tipo do processo ao qual este sub-processo está relacionado, o código deste novo sub-processo, o nome, o caminho do *log* deste sub-processo e os parâmetros. A tela ainda apresenta um campo que contém a data da última execução deste sub-processo e uma breve mensagem referente a execução dele.

No campo “Caminho do *log*” está a estrutura e o nome do *log* de cada sub-processo. Clicando neste nome, o arquivo de *log* de cada processo é apresentado; para isto, o administrador deverá sempre informar o caminho de *logs* na estrutura padrão, pré-determinada pela ferramenta e ainda, criar um *link* abaixo da configuração do *web-server* que aponte para esta estrutura.

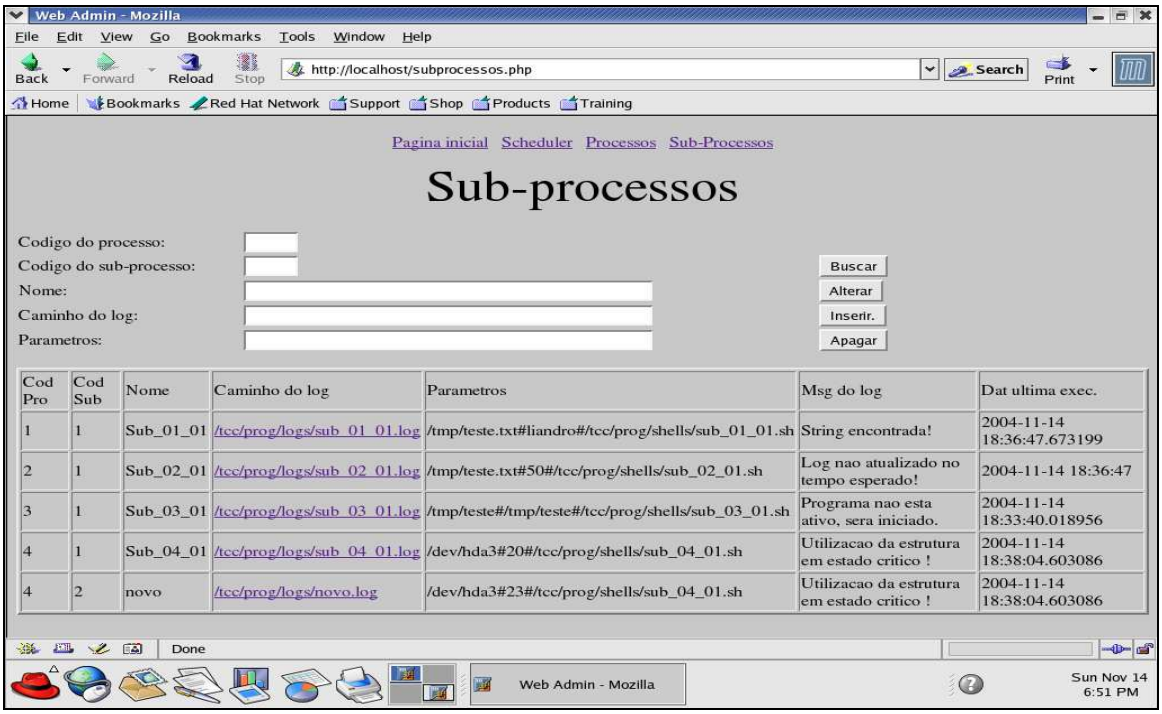

Figura 12 - Tela de configuração e monitoração dos sub-processos

O exemplo exibido na figura 13 mostra uma mensagem gerada a cada 20 segundos. Este intervalo de tempo deve ser previamente cadastrado na tela de configuração do escalonador:

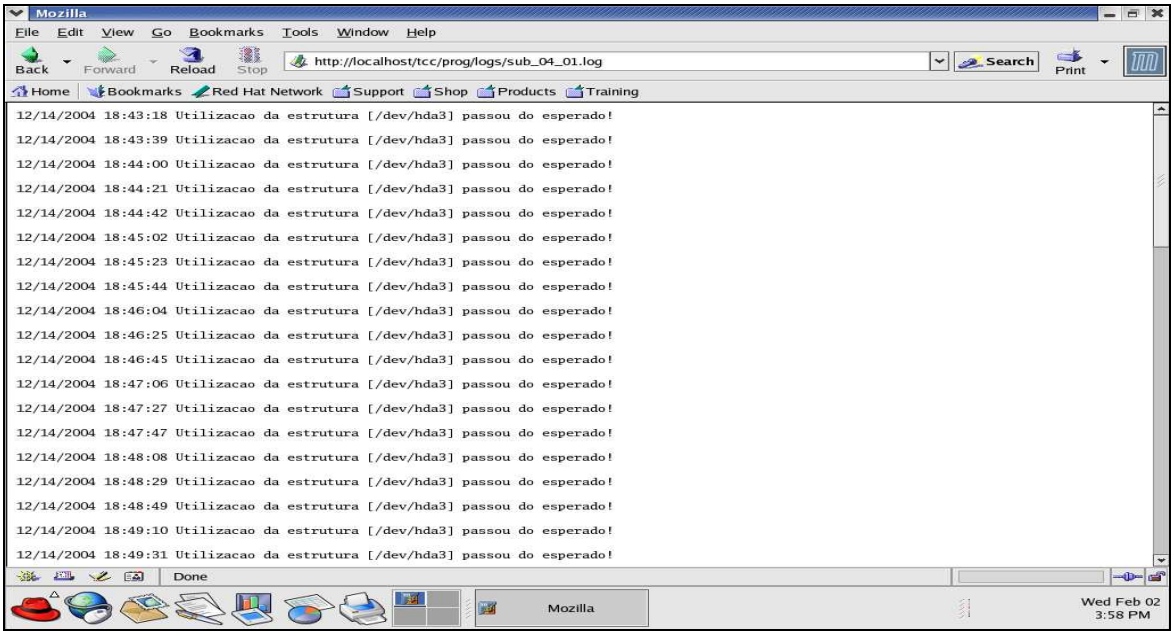

Figura 13 - Log de subprocesso

5 ESTUDO DE CASO

Abaixo está descrito um teste para cada situação que a ferramenta monitora.

No teste de busca por uma expressão de erro dentro de um arquivo de *log*, foi criado o arquivo “log.log” (vazio) na estrutura /testes. Depois disso foram cadastradas na ferramenta, as informações referentes a este arquivo: o nome do arquivo de *log*, a expressão que deveria ser procurada (“FATAL_ERROR”) e a ação a ser tomada (“/testes/programa.sh”) caso a expressão fosse encontrada, como pode ser visto na figura 14:

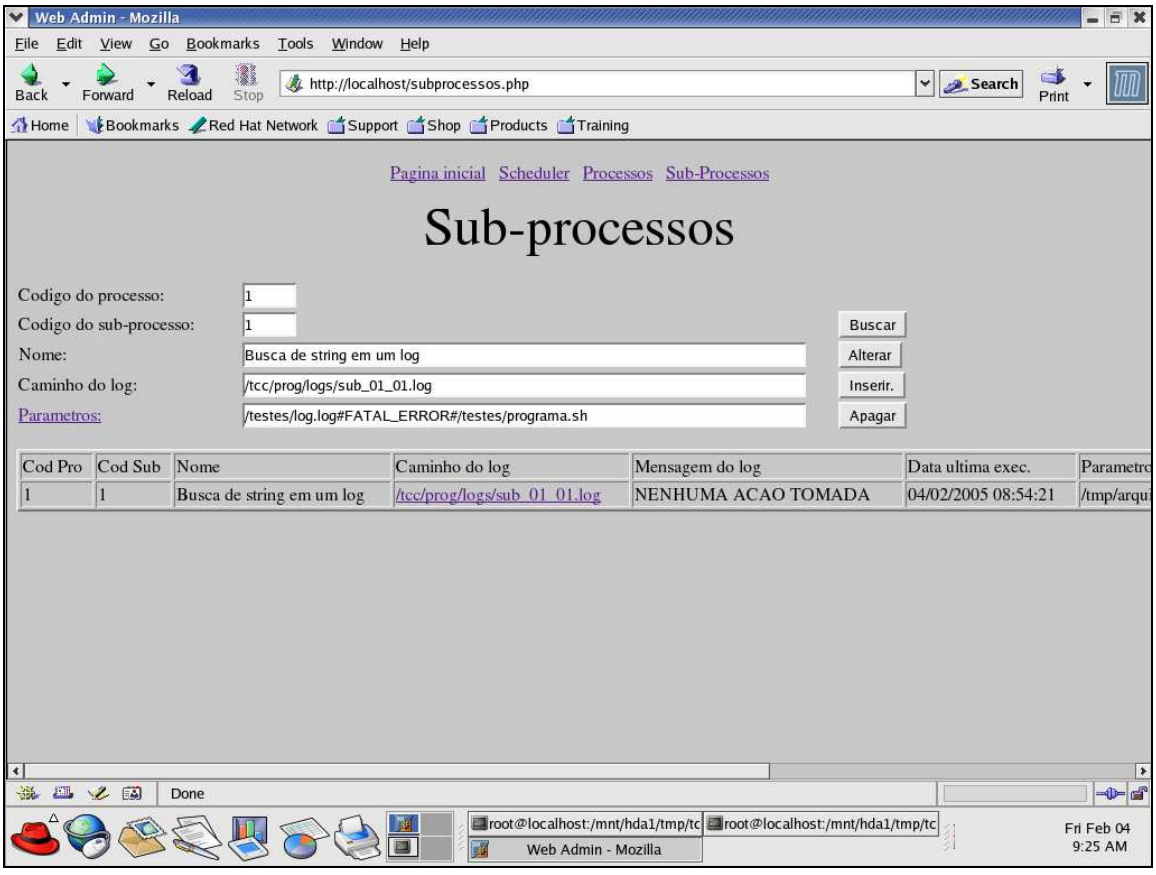

Figura 14 - Cadastro para o teste 1

Após isto, o escalonador foi iniciado e passou a verificar o arquivo /testes/log.log. Como o arquivo estava vazio, nada foi feito pelo escalonador. Assim que a expressão cadastrada na ferramenta foi inserida no arquivo, o escalonador disparou a ação associada a este sub_processo e atualizou a base de dados com as informações deste evento.

A figura 15 mostra os estados do arquivo log.log antes e após a atualização com a expressão FATAL_ERROR:

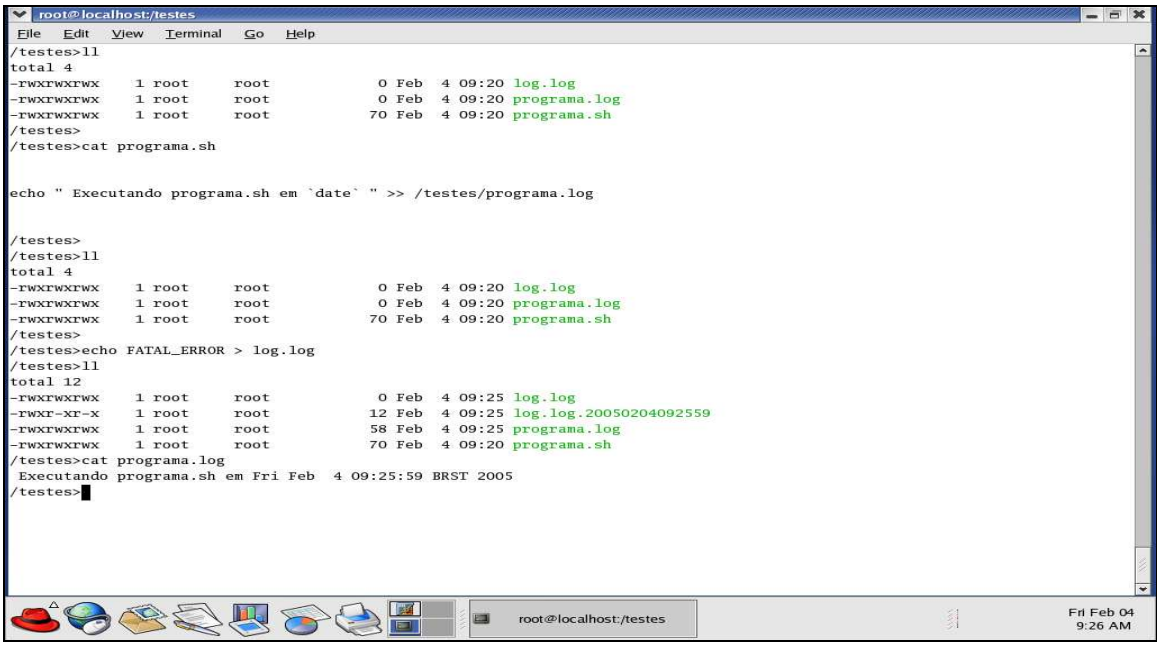

Figura 15 - Arquivos atualizados no teste 1

A figura 16 mostra a tela do sistema que o administrador visualiza, com as informações referentes a esta monitoração já atualizadas:

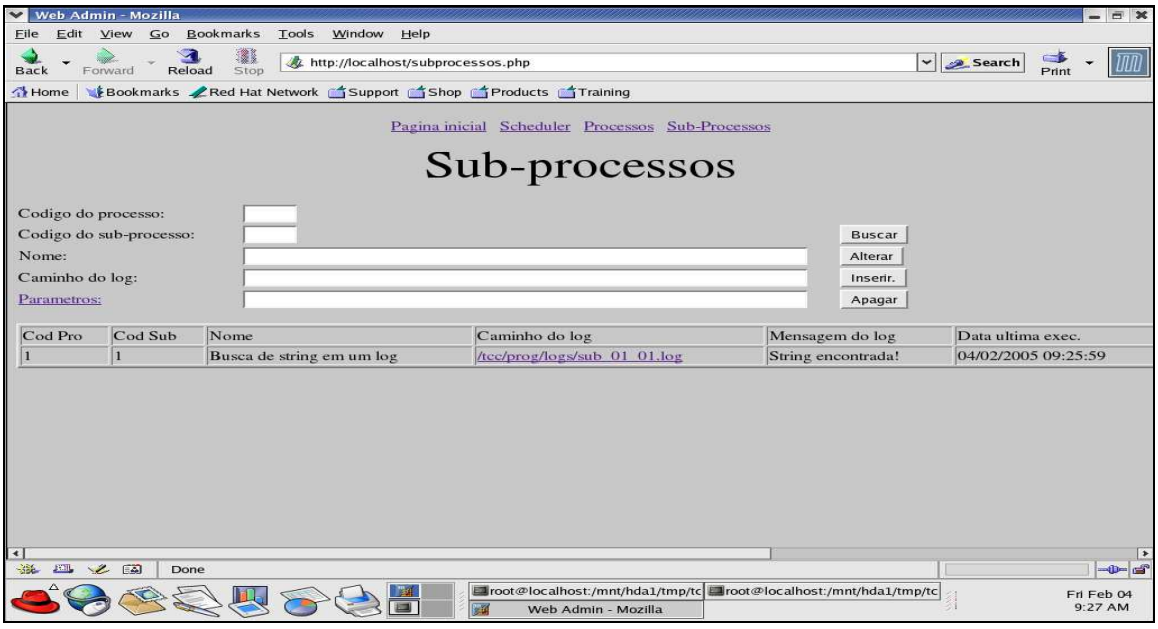

Figura 16 - Informações do teste 1

Para o segundo teste, verificação da data da última atualização de um arquivo de *log*, foi criado o arquivo verificar_data.txt na estrutura /testes. As informações cadastradas para este evento foram as seguintes: nome do arquivo a ser monitorado: /testes/verificar_data.txt; número de segundos que se ultrapassado, indicava que o processo que atualiza este arquivo está com algum problema: 35 ; ação a ser tomada caso o número de segundos informado seja ultrapassado: /testes/programa.sh, conforme figura 17.

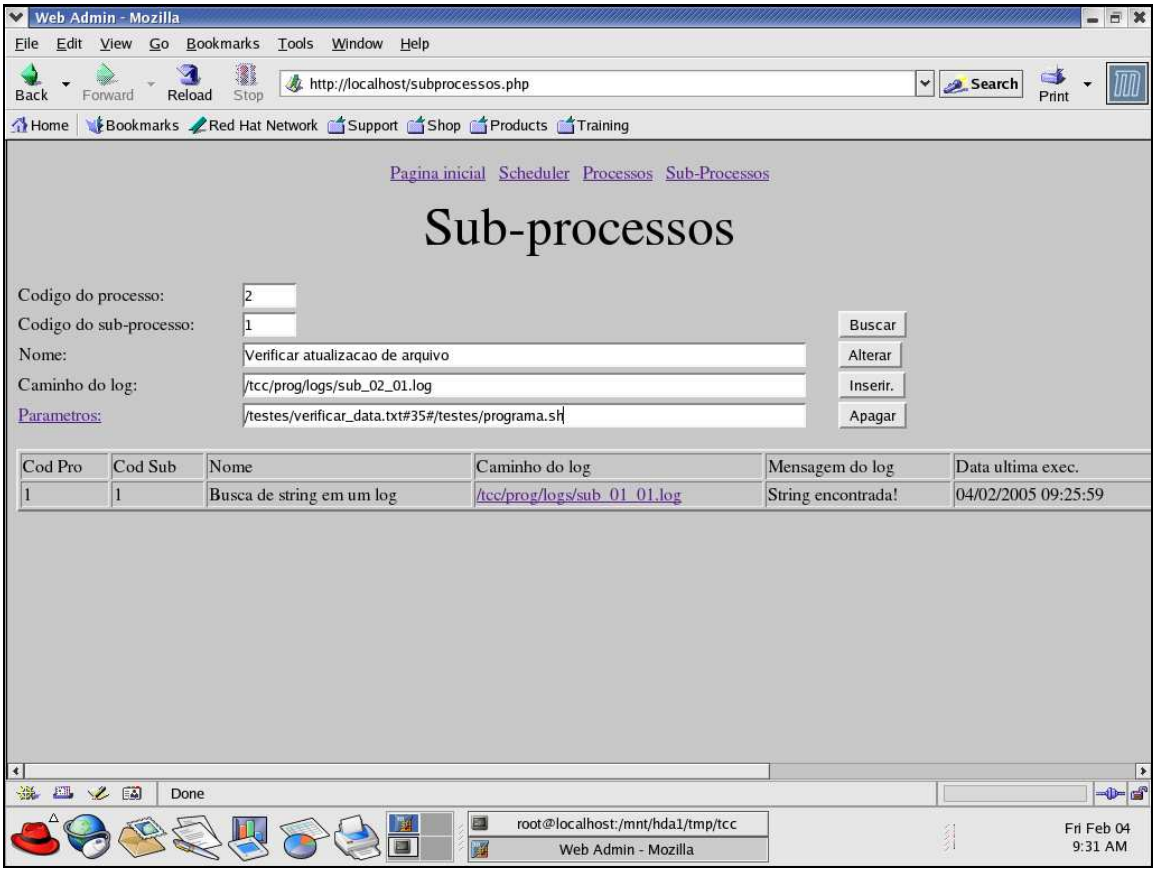

Figura 17 - Cadastro para o teste 2

O escalonador foi iniciado e nada foi feito até que se passaram 35 segundos do momento da última atualização do arquivo /testes/verificar_data.txt. Quando este tempo foi ultrapassado, o escalonador disparou o evento /testes/programa.sh, conforme figura 18:

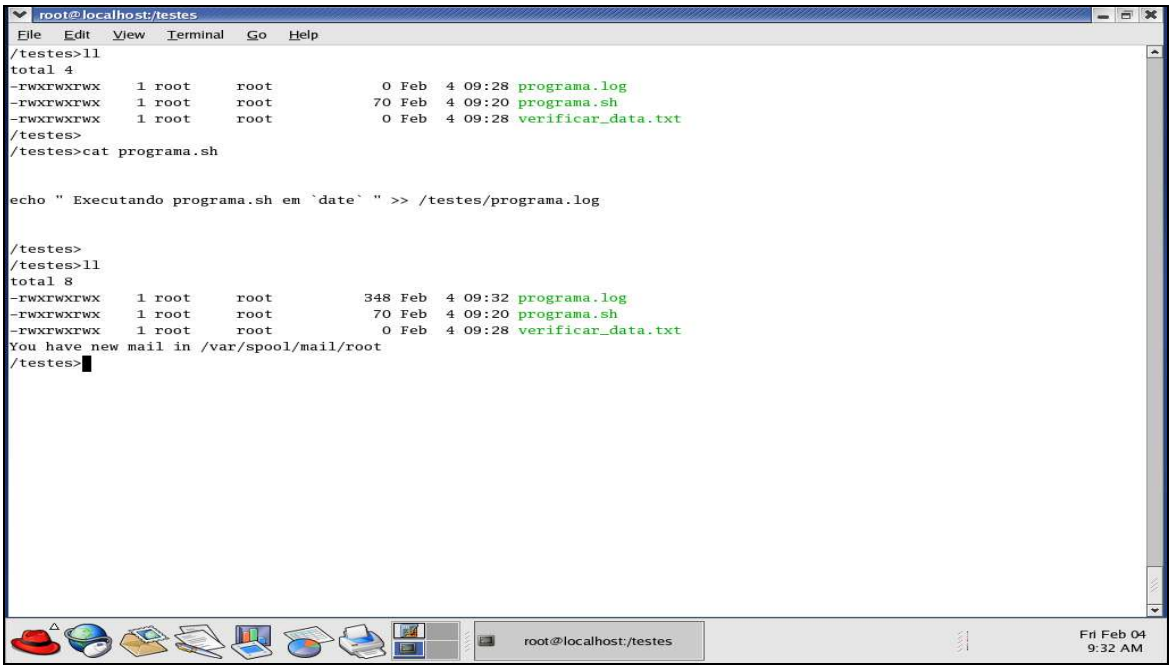

Figura 18 - Arquivos atualizados no teste 2

Depois disso, o escalonador atualizou a base de dados com as informações deste evento para que o administrador possa acompanhar o processo. Isto é visto na figura 19:

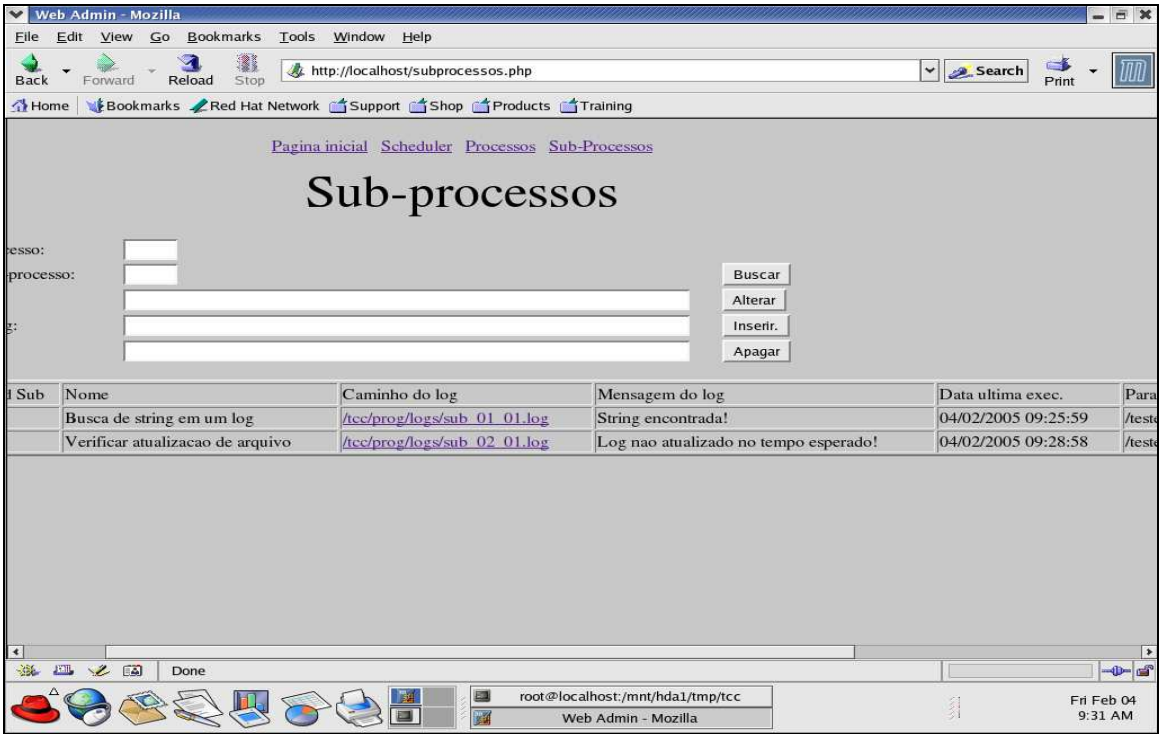

Figura 19 - Informações do teste 2

No terceiro teste, o escalonador é estimulado a procurar por um processo ativo no sistema operacional. Para este teste foi criado um programa executável em C, exibido na figura 20, (compilado a partir do processo_ativo.c) que nada faz além de ficar ativo para que seja monitorado.

```
/testes>ll
total 16
-rwxr-xr-x    1 root     root        11551 Feb  4 09:35 processo_ativo
-rwxr-xr-x    1 root     root           59 Feb  4 09:35 processo_ativo.c
/testes>
/testes>
/testes>cat processo_ativo.c


#include <stdio.h>

void main()
{
 system("sleep 30");
}
/testes>_
```

Figura 20 - Teste 3 - processo_ativo.c

Foram cadastradas na ferramenta as informações para a monitoração deste processo: o nome do processo que deveria ser monitorado: o executável “processo_ativo”; e a ação que deveria ser tomada caso o processo não estivesse ativo “/testes/processo_ativo” (isto fará com que ele seja novamente iniciado) conforme figura 21:

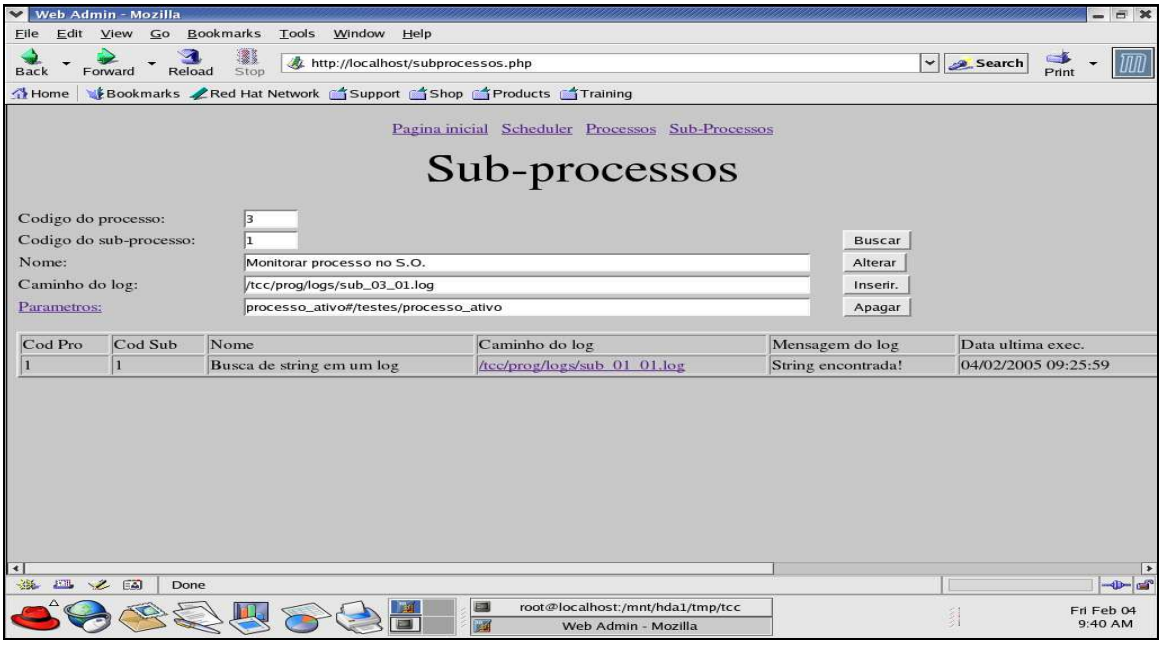

Figura 21 - Cadastro para teste 3

O processo criado em C foi disparado e o escalonador foi iniciado. Enquanto o processo em C estava ativo, o escalonador nada fez. Assim que o processo “processo_ativo” foi finalizado, o escalonador identificou que ele não estava ativo e neste instante, disparou a ação associada a este programa deixando-o novamente ativo e atualizou a base com as informações deste evento, conforme figura 22:

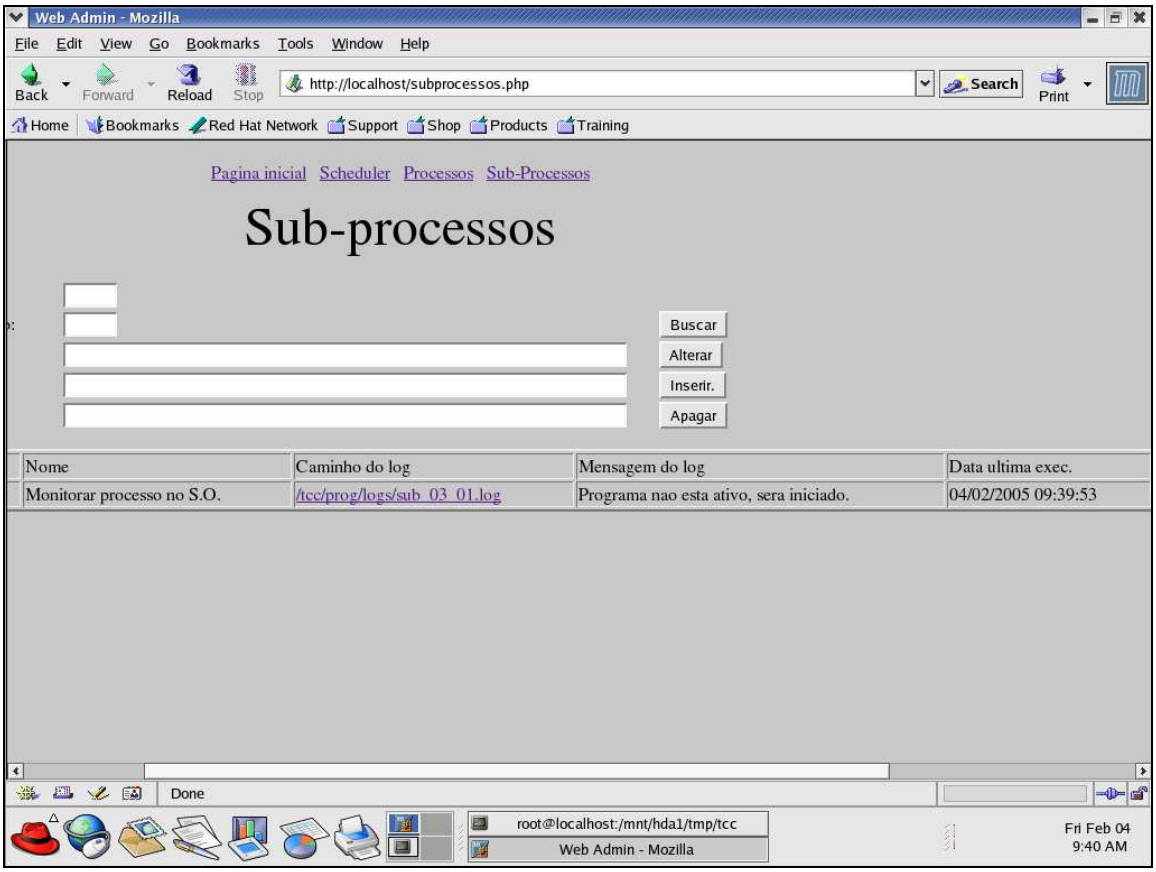

Figura 22- Informações do teste 3

O quarto teste consistiu em monitorar o percentual de utilização de uma estrutura física. Para este teste foram cadastradas na ferramenta duas situações: uma onde o percentual crítico é maior do que o real utilizado e uma onde o percentual crítico é inferior ao utilizado, para forçar o disparo do processo. Para isto, foram cadastradas as seguintes informações: nome da estrutura a ser monitorada: /dev/hda3; percentual crítico de 81% para o primeiro caso e 79% para o segundo caso; ação a ser tomada caso o percentual de utilização seja excedido: /testes/programa.sh, como pode ser visto na figura 23:

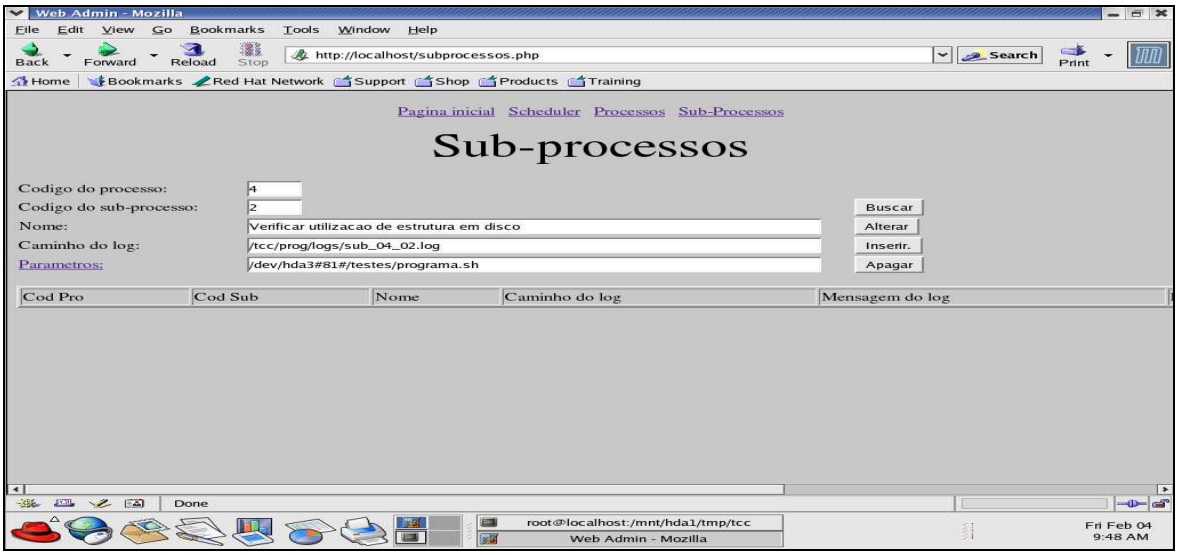

Figura 23 - Cadastro para o teste 4

O escalonador foi iniciado e passou a monitorar o percentual de utilização desta estrutura, que estava em 80%. No cadastro feito onde o percentual crítico era maior do que o percentual utilizado, o escalonador nada fez. Já para o percentual crítico cadastrado que estava menor do que o utilizado, a ferramenta disparou a ação associada a este evento, /testes/programa.sh e atualizou a base com as informações deste evento, isto é mostrado na figura 24:

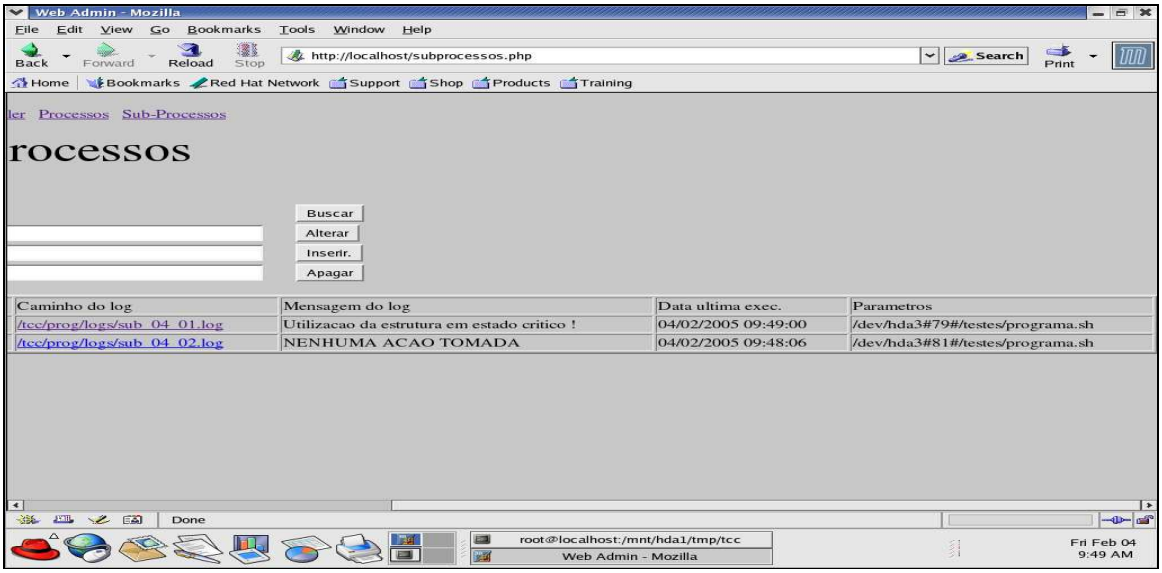

Figura 24 - Informações do teste 4

No quinto teste, o escalonador dispara um processo com os parâmetros informados pelo usuário. Para este teste, foram cadastradas as informações a seguir: nome do processo que será disparado a cada ciclo: /testes/programa.sh ; e os parâmetros deste processo “arquivo1 arquivo2 arquivo3”, conforme figura 25:

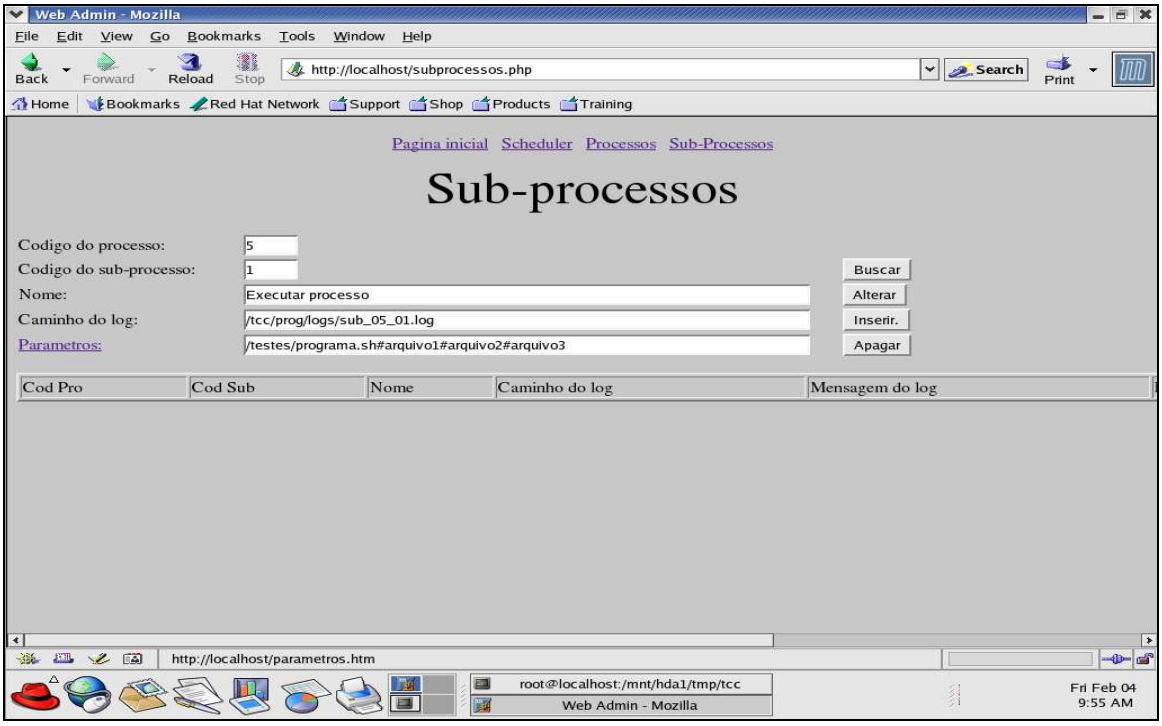

Figura 25 - Cadastro para teste 5

O programa /testes/programa.sh era apenas um exemplo e criava um arquivo contendo o nome dos parâmetros recebidos com uma extensão “.txt” . No final do ciclo do escalonador, os arquivos criados estavam com os nomes “arquivo1.txt arquivo2.txt arquivo3.txt”, informando que o processo tinha sido disparado corretamente, isto é visto na figura 26 exibida abaixo:

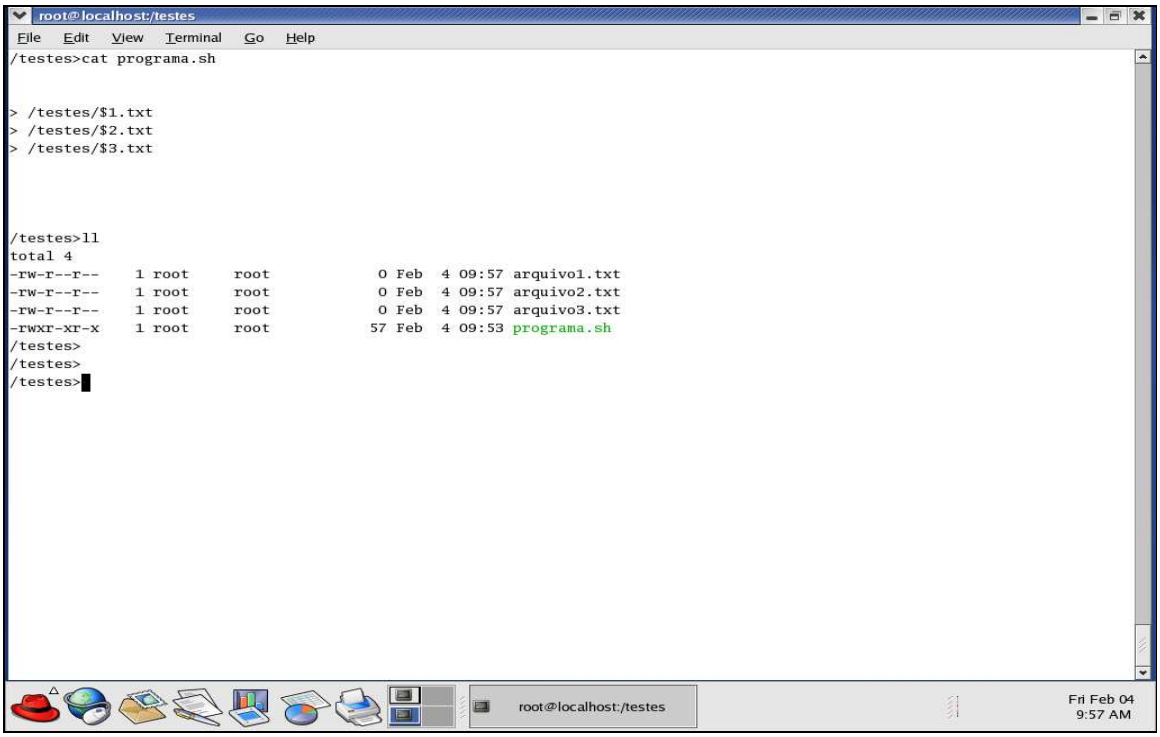

Figura 26 - Arquivos criados no teste 5

5.1 FORMA DE PASSAGEM DOS PARÂMETROS

Os parâmetros cadastrados devem ser sempre informados separados pelo símbolo de sustenido “#” e devem estar sempre na mesma ordem, para que a ferramenta trabalhe como o proposto. Segue abaixo a explicação do cadastro dos parâmetros dos sub-processos, para cada tipo de processo pré-definido na ferramenta administrativa:

a) código 1: Buscar uma expressão em algum *log*:
   - o primeiro parâmetro é a estrutura completa com o nome do arquivo que deverá ser monitorado,
   - o segundo parâmetro é a mensagem que deverá ser procurada neste arquivo,
   - o terceiro parâmetro é a ação que deve ser tomada caso a mensagem procurada no arquivo seja encontrada;

b) código 2: Verificar data da última atualização de um arquivo de *log:*

- o primeiro parâmetro é a estrutura completa com o nome do arquivo de *log* que deverá ser monitorado,
- o segundo parâmetro é o tempo em segundos que será comparado pela ferramenta administrativa, com o tempo obtido entre a diferença de tempo da data atual menos a data da última atualização do arquivo,
- o terceiro parâmetro é a ação que deverá ser tomada caso o tempo obtido pela ferramenta seja maior do que o tempo informado;

c) código 3: Verificar se processo está ativo:
   - o primeiro parâmetro é o nome do processo que deverá ser monitorado,
   - o segundo parâmetro é a ação que deverá ser executada caso o processo monitorado não encontre-se ativo;

d) código 4: Verificação de utilização de espaço em disco:
   - o primeiro parâmetro informa o nome da estrutura a ser monitorada,
   - o segundo parâmetro informa o percentual crítico de utilização desta estrutura,
   - o terceiro parâmetro informa a ação que será executada caso a ferramenta identifique que o percentual de utilização da estrutura seja igual ou maior do que o percentual de utilização crítico que foi cadastrado para esta estrutura;

e) demais: demais processos cadastrados :
   - o primeiro parâmetro informa o processo que deverá ser disparado,
   - o segundo parâmetro deverá conter todos os parâmetros que serão passados para o processo disparado. Aqui, os parâmetros deverão estar separados por espaços.

5.2 RESULTADOS E DISCUSSÃO

Através dos testes feitos no ambiente criado, conforme demonstrado neste capítulo, a ferramenta mostrou-se eficaz nas soluções apresentadas. A ferramenta não apresentou

problemas, foi rápida para identificar as situações e tomar as devidas providências para cada tipo de situação que deveria estar monitorando.

Foi verificado que o número de sub-processos cadastrados influencia no tempo de resposta da ferramenta. Isto porque a verificação é feita individualmente para cada sub-processo; ou seja, se existirem 60 sub-processos cadastrados e cada um deles levar 1 segundo para ser executado, o último sub-processo ficará esperando por 60 segundos para iniciar sua execução. Caso este sub-processo precise fazer uma verificação em um intervalo de tempo menor do que 60 segundos, a ferramenta não trabalhará como o esperado.

6 CONCLUSÕES

A ferramenta mostrou-se estável e não apresentou erros para acessar arquivos de *log* e também para disparar processos dos usuários pois está configurada para executar na conta do administrador do sistema "*root*". O tempo de resposta foi bom; porém, pode apresentar problemas caso o número de verificações fique muito extenso.

Procurando melhor aproveitamento do tempo dos administradores de ambientes computacionais, este trabalho visou o desenvolvimento de uma ferramenta que pudesse monitorar estes ambientes sem que se fizesse necessária uma intervenção permanente do administrador. O uso da ferramenta resulta no ganho de tempo, por parte do administrador do ambiente, por ele não precisar mais ser envolvido para efetuar acertos no ambiente nas situações tratadas pela ferramenta. Além do ganho do tempo por parte do administrador, a ferramenta atua no ambiente computacional muitas vezes sem ser percebida, pois toma as ações no menor espaço de tempo possível entre o acontecimento dos problemas e a aplicação da solução pré-determinada.

6.1 EXTENSÕES

Como sugestão para trabalhos futuros, poderia ser feita a divisão da ferramenta em módulos, um para cada tipo de processo. Estes módulos deveriam ser gerenciados por um programa principal e estariam trabalhando independentemente para submeter seus sub-processos. Isto estaria reduzindo o tempo de espera na execução de cada sub-processo, o que pode se tornar um problema caso este número se torne muito grande.

Poderia ainda ser feita uma configuração de estado para cada tipo de sub-processo, para saber se ele deve ou não ser monitorado pelo escalonador em cada ciclo. Isto evitaria a necessidade de remover um sub-processo das tabelas do sistema quando ele não deve ser monitorado. Seria ganho também um histórico de todos sub-processos que já foram monitorados. Hoje nas tabelas do sistema aparecem apenas os sub-processos que estão, efetivamente, sendo monitorados.

Outra melhoria é com relação a conta que está submetendo os sub-processos monitorados. Hoje a ferramenta está configurada para submeter todos processos com a conta *root*. Isto poderia ser alterado para que cada sub-processo possa um campo que informe qual a conta que deve ser utilizada para submeter o sub-processo. Apenas o processo principal estaria sendo executado na conta *root* e cada sub-processo ficaria em uma conta pré-determinada.

REFERÊNCIAS BIBLIOGRÁFICAS

ABLES, Robert King. **The keys to successful UNIX system management.** New Jersey: Prentice-Hall, 1995.

CHRISTIAN, Kaare. **Tópicos avançados do sistema UNIX**. Rio de Janeiro: Campus, 1987.

DANESH, Arman. **Dominando o LINUX:** a bíblia. São Paulo: Makron Books do Brasil Editora Ltda, 2000.

DAVIS, William S. **Sistemas operacionais**. Rio de Janeiro: Campus, 1990.

DEITEL,Harvey M. **Operating System**s. Estados Unidos da América: Addison-Wesley Publishing Company, 1990.

DIMOV, Jordam. **Segurança em programação PHP**. [s.l.], [2004?]. Disponível em: <http://www.superphp.com.br/artigos/?id=8> . Acesso em: 15 nov. 2004.

HANSEN, Augie. **Salvo pelo... UNIX**. São Paulo: Érica,1995

MEDEIROS, Eduardo Luis de. **O que é PHP ?** . [s.l.], [2004?]. Disponível em: <http://www.superphp.com.br/artigos/index.php?id=2> . Acesso em: 15 nov. 2004.

NEMETH, Evi**. Manual de administração do sistema UNIX**. Porto Alegre: Bookman, 2002.

NEVES, Denise Lemes Fernandes. **PostgreSQL:** conceitos e aplicações. São Paulo: Érica, 2002.

PEREIRA NETO, Álvaro. **PostgreSQL:** técnicas avançadas. São Paulo: Érica, 2003.

RAMALHO, José Antônio Alves. **HTML 4:** teoria e prática. São Paulo: Berkeley, 2001.

SCHILDT, Herbert. **C completo e total**. São Paulo: Makron Books do Brasil Editora Ltda, 1996.

SANTOS, Fábio. **UNIX.** [s.l.], [2004 ?]. Disponível em: <http://orbita.starmedia.com/~fabio-rs/unix.htm>. Acesso em: 03 out. 2004.

SILVA, Osmar J. **Dynamic HTML**. São Paulo: Érica, 2001.

SOARES, Walace. **PHP4 e MySql**. São Paulo: Érica, 2002.

TANENBAUM, Andrew S.; WOODHULL, Albert S**. Sistemas operacionais:** projeto e implementação. Porto Alegre: Artes Médicas Sul Ltda, 2000.