JOSÉ PAULO DA SILVA

Gerência de Processos: Monitoramento de Processos Inativos de Um Sistema Proprietário.

Monografia de Pós-Graduação apresentada ao Departamento de Ciência da Computação para obtenção do título de Especialista em Administração de Redes Linux.

Orientador

Prof. Arlindo Follador Neto

LAVRAS
MINAS GERAIS – BRASIL
2009

JOSÉ PAULO DA SILVA

Gerência de Processos: Monitoramento de Processos Inativos de Um Sistema Proprietário.

Monografia de Pós-Graduação apresentada ao Departamento de Ciência da Computação para obtenção do título de Especialista em Administração de Redes Linux.

APROVADA em ___ de Novembro de _____

Prof._____

Prof._____

Prof. Arlindo Follador Neto

(Orientador)

LAVRAS

MINAS GERAIS – BRASIL

2009

Dedico esta monografia a meus pais e aos meus companheiros de trabalho.

Agradecimentos

Agradeço a todos que de maneira direta ou indireta me apoiaram no desenvolvimento deste trabalho e na conclusão desta especialização.

Sumário

Lista de Figuras

Lista de Tabelas

RESUMO

Este trabalho apresenta um estudo de caso sobre gerenciamento de processos inativos no Linux originados por um sistema proprietário. A motivação para o desenvolvimento deste trabalho surgiu devido a problemas de falta de disponibilidade no acesso a este sistema. Há um limite máximo de acessos permitido o que ocasiona a indisponibilidade de acesso ao sistema após este limite ser excedido. O êxito deste estudo é determinante para aumentar a produtividade dos usuários, do administrador de redes, além de reduzir custos de aquisição de licenças de software adicionais. A metodologia utilizada foi a criação de um aplicativo em Shell Bash que utiliza banco de dados textual com acesso através de interface amigável no Shell do Linux. O aplicativo desenvolvido aumentou a disponibilidade da aplicação visto que conexões ociosas foram removidas

Palavras-Chave: Processos; Licenças; Gerenciamento; Ocioso.

1 – Introdução

Um sistema operacional é um componente essencial de qualquer computador e pode ser definido como um programa intermediário entre o usuário e o hardware. Uma das funções do sistema operacional é fornecer meios para que um programa seja carregado para a memória e executado. Em sistemas operacionais de tempo compartilhado, a capacidade de processamento da CPU é dividida entre os programas em execução de acordo com uma prioridade definida.

Determinados programas são caracterizados por terem restrição quanto ao número de usuários que podem utilizá-lo em um mesmo instante. Esta restrição prejudica a disponibilidade de acesso ao sistema quando o número de usuários conectados ultrapassa a quantidade de acessos permitida. Esta monografia objetiva criar uma solução para gerenciar programas com limite de acesso no sistema operacional Linux. Este gerenciamento poderá será feito através da interrupção dos processos[1] que estão ociosos por determinado tempo, liberando o acesso para outros usuários.

A motivação para o desenvolvimento deste trabalho deu-se por problemas na disponibilidade de acesso em um sistema proprietário de uma empresa, em um ambiente Linux. A licença de uso do sistema permitia o acesso a um número determinado de terminais, de acordo com o número de licenças contratadas pela empresa. A indisponibilidade de acesso ocorria quando o número de usuários conectados atingia o número de acessos limite, ocasionando em perda de produtividade dos usuários que tivessem o acesso negado. Na elaboração do estudo de caso percebeu-se que o motivo principal da indisponibilidade de acesso ocorria devido à terminais conectados ao sistema

1 Processo é um programa em execução no sistema operacional Linux.

ociosos por um tempo significativo. Esta ociosidade impedia o acesso de usuários que necessitavam utilizar o sistema. Este estudo de caso propõe uma solução que visa gerenciar com eficiência os terminais ociosos conectados ao sistema. O objetivo principal é desconectar estes terminais, liberando o acesso para outros usuários.

Duas hipóteses foram estudadas para o desenvolvimento deste trabalho. A primeira hipótese trata-se do uso de uma variável especial do Linux chamada TMOUT, que recebe o valor em segundos do tempo máximo permitido de ociosidade de uma sessão no Linux. A variável deverá ser configurada no arquivo .bashrc, no diretório pessoal de cada usuário. A segunda hipótese trata-se da criação de um programa em Shell Bash que fará o monitoramento de cada usuário através de um cadastro definido anteriormente, utilizando banco de dados textual e interfaces amigáveis através do Dialog[2].

Este trabalho encontra-se organizado como se segue: O capítulo 2 apresenta conceitos sobre licenças de software , o capítulo 3 são apresentados conceitos sobre processos no Linux; o capítulo 4 descreve formas de implementação de bancos de dados textuais; O capítulo 5 apresenta o aplicativo Dialog; o capítulo 6 apresenta o Aplicativo GPI resultante deste trabalho; o Capítulo 7 apresenta os testes realizados e resultados obtidos; o Capítulo 8 apresenta a conclusão do trabalho.

2 Programa de console (modo texto) que desenha janelas na tela, com menus, caixas de texto e botões.

2 – Licenças de Software

Licenças de software são condições definidas para que o usuário possa utilizar um software. São equivalentes a um contrato, pois definem formas de pagamento, utilização, meios para atualização e condições para suporte técnico. De acordo com a legislação de direitos autorais (regida pela lei 9609/98) ou o que for definido pelas partes, podem incidir regras sobre o uso, a modificação, distribuição ou cópia do software.

Na aquisição de uma Licença de Software, o usuário não detêm os direitos de sua propriedade. A propriedade pertence ao desenvolvedor, que obtêm lucros com a venda da licença e com o suporte técnico. A legislação de direitos autorais prevê a permissão de cópia de segurança para uso pessoal do usuário.

Existem outras modalidades de Licenças de Software que não se enquadram na legislação de direitos autorais, pois pregam a livre distribuição, modificação, cópia e utilização do programa. Dentre elas pode-se citar a GPL[3] e a Open Source[4] que disponibilizam vários softwares de várias categorias para download. Esta modalidade de licença é conceituada como Software Livre.

2.1 – Licenças de Software Livre

O conceito de Software Livre surgiu inicialmente através da criação do projeto GNU[5]. Iniciado por Richard Stallman em 1984, tinha o objetivo de criar um sistema operacional totalmente livre, portanto permitindo que qualquer usuário tivesse direito de usar, modificar e redistribuir o programa desde que

3 http://www.gnu.org GNU General Public Licence ou GPL (Licença Pública Geral) é a designação da licença para software livre idealizada por Richard Stallman.
4 http://www.opensource.org
5 http://www.gnu.org

fossem respeitados os direitos do autor. O projeto GNU visava ter total compatibilidade com o sistema UNIX, entretanto, não seria utilizado nenhum código fonte do UNIX.

Entre as licenças de software livre, destaca-se a GPL[6] , idealizada por Richard Stallman, que possui a maior parte dos projetos de software livre desenvolvidos, principalmente devido à sua adoção pelo Linux[7]. A GPL é baseada em quatro liberdades:

- Executar o programa para qualquer fim;
- Estudar o funcionamento do programa, e adaptá-lo às suas necessidades;
- Redistribuir cópias com o objetivo de servir para outras pessoas;
- Aperfeiçoar o programa, liberando suas melhorias para outras pessoas, de modo que a comunidade se beneficie das melhorias.

Outras Licenças de Software Livre possuem regras diferentes da GPL quanto à liberdade do usuário. A licença BSD[8] por exemplo, diferencia-se da GPL principalmente em relação à obrigatoriedade de distribuição do código fonte do programa. Desta forma, usuários podem usufruir de códigos fontes de programas baseados em licença BSD e após algumas melhorias torná-lo um software proprietário.

2.2– Licenças de Software Proprietário

O Software Proprietário define-se pela restrição de cópia, modificação ou redistribuição pelo criador ou distribuidor. Desta forma, o usuário tem o direito de apenas utilizar o programa, sem possuir sua propriedade.

6 GNU General Public Licence (Licença Publica Geral) ou simplesmente GPL
7 Kernel livre escrito em 1991 por Linus Torvalds baseado no Unix
8 BSD (Berkley Software Distribuition), licença utilizada incialmente em sistemas operacionais BSD (sistema derivado do UNIX).

A Licença de Software Proprietário define regras que regulam como o usuário deverá utilizar o programa, quais os direitos e deveres que ele deverá seguir após a aquisição de uma licença de uso. Ela é caracterizada no conceito jurídico como *Copyright,* dando ao criador de uma obra a autoria dos direitos exclusivos, geralmente por um tempo determinado.

O custo de um Software Proprietário para as empresas torna-se muito elevado, pois em grande parte, deverá ser adquirida uma licença para cada máquina utilizada com o programa. O controle das licenças, em grande parte, é feito pelo próprio programa, sendo que, após o número de usuários conectados atingir o número de licenças contratadas, não será mais permitida a conexão ao programa.

O programa objeto de estudo deste trabalho possui a licença de software proprietário, portanto está sujeito a todas as restrições impostas por esta modalidade de licença de uso.

Independente do tipo de licença de um software, quando ele é executado é conceituado como um processo. O capítulo 3 mostra como os processos são tratados no Linux.

3 – Processos no Linux

Nos sistemas operacionais, um processo pode ser definido como um programa em execução. A gerência dos processos no Linux[9] é uma tarefa do kernel[10], que utiliza um escalonador de processos[11] para definir qual processo será executado pela CPU[12]. Cada processo executado no Linux é representado por uma estrutura chamada Process Control Block (PCB) que armazena informações vinculadas ao processo específico. O PCB armazena informações como: estado do processo, contador de programa, registradores da CPU, escalonamento de CPU, gerência de memória, contabilização e informação de E/S. Os estados possíveis para um processo são os seguintes (Silberchatz, 2004):

- Novo: quando o processo está sendo criado;
- Executando: as instruções estão sendo executadas;
- Esperando: o processo está esperando pela ocorrência de algum evento (como o término de entrada e saída ou recepção de um sinal);
- Pronto: o processo está esperando para ser designado a um processador;
- Terminado: o processo terminou sua execução;

A Tabela 3.1 apresenta detalhes das informações contidas no PCB:

Tabela 3.1 – Informações do PCB

Item	Descrição
Estado do processo	Novo, executando, esperando, pronto, terminado;

9 Sistema operacional desenvolvido a partir do kernel Linux.
10 No Linux, o kernel é o núcleo do sistema operacional (SO), que fornece sistema de arquivos, escalonamento de cpu, gerência de memória e outras funções do SO através de chamadas de sistema.
11 O escalonador de processos divide o tempo do processador entre os processos.
12 CPU (Central Processing Unit) ou Unidade Central de Processamento.

Tabela 3.1 – Informações do PCB (continuação)

Item	Descrição
Contador de programa	Endereço da próxima instrução a ser executada pelo processo;
Registradores da CPU	Acumuladores, registradores de índice e uso, ponteiros de pilha;
Escalonamento CPU	Inclui prioridade de processo, ponteiros para filas, escalonamento, além de outros parâmetros;
Gerência de memória	Inclui valores dos registradores base e limite, as tabelas de páginas ou tabelas de segmentos;
Contabilização	Inclui a quantidade de tempo de CPU e de tempo real utilizado, limites de tempo, registros de contabilidade, número de jobs ou processos, etc;
Estado de E/S	Inclui lista de dispositivos de E/S;

De acordo com (TANEMBAUM,1997):

> *Em um sistema de multiprogramação, a CPU também alterna de um programa para outro, executando cada um por dezenas ou centenas de milissegundos. Enquanto, estritamente falando, em qualquer instante de tempo, a CPU está executando só um programa, no curso de 1 segundo, ela pode funcionar para vários programas, dando aos usuários a ilusão de paralelismo.*

Nos sistemas multiprogramados, enquanto um processo aguarda a ocorrência de um evento externo à CPU, esta pode atender a outro processo na fila de espera para execução. A Figura 3.1 descreve este funcionamento, nela pode-se observar que em determinado momento na linha do tempo, apenas um

processo permanece em execução na CPU. Esta velocidade de alternâncias de execução dá ao usuário a impressão que os processos estão executando simultaneamente.

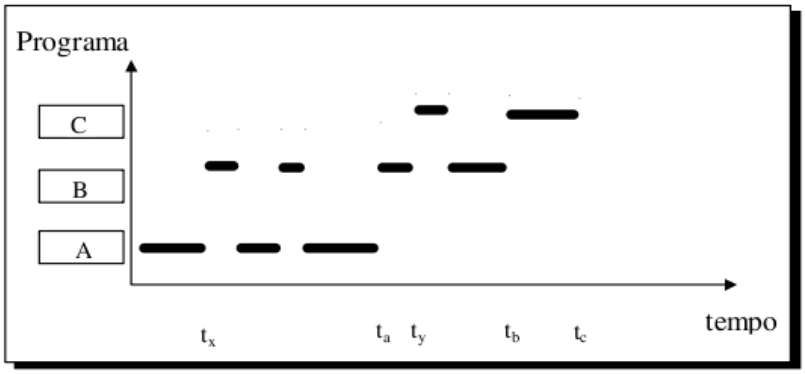

Figura 3.1 – Processos em um sistema multiprogramado

No Linux, o kernel armazena os processos criados em uma lista circular duplamente encadeada. Esta lista é denominada lista de tarefas, onde cada elemento da lista é um descritor de processo do tipo struct task_struct. Nesta estrutura estão todas as informações referentes aos processos, portanto, o PCB no Linux é implementado nesta estrutura.

A escolha de qual processo será o próximo a ser executado pela CPU dependerá do escalonador de processos. Ele visa dar mais eficiência nas tarefas do processador para que a todo instante um processo esteja em execução. Para que o escalonador de processos funcione é necessário ter uma política de escalonamento. Através da política utilizada, os processos podem ser classificados em I/O-bound ou CPU-bound.

Os processos I/O-bound gastam a maior parte do seu tempo submetidos

e esperando por requisições I/O (entrada e saída). Os processos CPU-bound gastam um tempo maior executando código. No Linux os processos I/O-bound tem maior prioridade para execução, proporcionando uma melhor resposta interativa com o usuário.

Apesar do Linux dar maior prioridade a processos I/O-bound, o usuário pode definir uma prioridade maior a um processo, objetivando influenciar o comportamento do escalonamento no sistema. Se um processo tem uma prioridade maior que um processo em execução, a CPU irá bloqueá-lo e executar o que tem maior prioridade. Um processo também pode ser bloqueado por algum evento externo, e dar preferência para o próximo da lista. A Figura 3.2 demonstra como funciona as alternâncias de processos na CPU.

O capítulo 3 apresenta conceitos sobre banco de dados textual que será utilizado neste trabalho para armazenar os processos que serão monitorados no sistema.

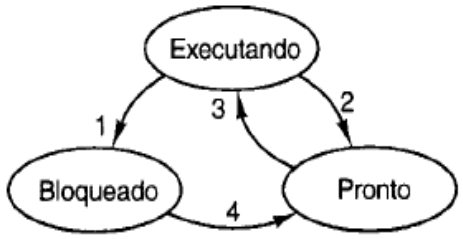

Figura 3.2 – Alternância de processos na CPU.

4 – Bancos de Dados Textuais

Banco de dados são estruturas organizadas que possibilitam armazenar informações para serem consultadas posteriormente. Em um sistema computacional, utilizar um banco de dados robusto para armazenar informações de programas de pequeno e médio porte é desnecessário. Uma alternativa que atende esta necessidade é o banco de dados textual.

Banco de Dados Textual define-se como uma estrutura simples que possibilita armazenar informações em arquivos texto, sem formatação ou imagens. Esta implementação utiliza os próprios recursos do sistema operacional para simular as funcionalidades básicas de um banco de dados tradicional. A Tabela 4.1 demonstra as vantagens e desvantagens da utilização deste tipo de banco de dados.

De acordo com a Tabela 4.1, percebe-se que a simplicidade de implementação é uma das maiores vantagens do banco de dados textual. Este tipo de banco de dados pode ser utilizado em vários tipos de aplicações. Alguns exemplos são descritos abaixo:

Tabela 4.1–Vantagens e Desvantagens do uso de Banco de Dados Textual

Vantagens	
Acesso fácil ao banco	Pode-se utilizar qualquer editor para fazer manutenção no banco e alterar dados;
Portabilidade	O banco poderá ser utilizado em qualquer sistema operacional;
Compactável	Possibilidade de compactação para que o arquivo fique com seu tamanho reduzido;
Simplicidade	Sua maior característica é a simplicidade da sua estrutura;

Tabela 4.1–Vantagens e Desvantagens do uso de Banco de Dados Textual (cont.)

Desvantagens	
Eficiência	Caso houver um crescimento do volume de dados, a velocidade de acesso é prejudicada;
Relacionamentos	Não tem relacionamentos entre os dados com outros arquivos, apesar de ser possível implementar;
Fragilidade	Está sujeito a acidentes que podem ocorrer com arquivos, como corrupção dos dados ou edição descuidada;

- Agenda de contatos pessoal;
- Catálogos de cds ou arquivos MP3;
- Controle de estoque simples;
- Arquivos de configuração de programas;
- Cadastro de usuários para fins específicos;

A Figura 4.1 demonstra o esquema de um Banco de dados Textual.

De acordo com (JARGAS, 2008):
O Banco de Dados Textual usa a própria hierarquia já existente no sistema de arquivos. Há diretórios que contêm arquivos, e arquivos que contêm texto. Cada arquivo funciona como uma tabela de um banco de dados relacional.

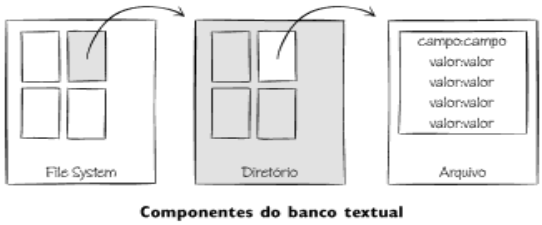

Figura 4.1 – Esquema de Banco de Dados Textual

De acordo com a Figura 4.1, o Banco de Dados Textual utiliza entidades já existentes no sistema de arquivos, como diretórios e arquivos. Desta forma, resta implementar a estrutura do arquivo texto. É necessário definir um padrão de armazenamento dos registros.

O CSV[13] é um padrão muito utilizado para definição de formatos de Banco de Dados Textual. A especificação deste formato é descrita a seguir:

- Um registro por linha;
- Os nomes dos campos ficam na primeira linha, as demais são os dados;
- Campos e dados separados por vírgulas;
- Os textos são colocados entre aspas e os números colocados diretamente;

A Figura 4.2 exibe um exemplo de arquivo texto que utiliza o padrão CSV. Maiores detalhes sobre a formatação de banco de dados textuais pode ser obtida em (JARGAS ,2008).

```
"Nome","Sobrenome","Apelido","E-mail","Fone","Celular"
"Maria","Cunha da Silva","maria","","","(41) 9999-1234"
"João","Almeida","joao","joao@email.com","3456-7890","9999-5678"
"","","zeca","zeca@coldmail.com","5432-9876",""
```

Figura 4.2 – Arquivo texto no padrão CSV.

Após a definição do padrão de arquivo a ser utilizado, é necessário ter um gerenciador que manipule os dados no arquivo.

A Figura 4.3 descreve o processo de comunicação do gerenciador com o banco de dados.

13 (Comma Separated Values) ou Valores separados por vírgulas.

Comunicação com o gerenciador do banco de dados

Figura 4.3 – Gerenciador de Banco de Dados

A seguir, o capítulo 4 descreve como construir interfaces gráficas amigáveis no Shell do Linux que podem ser utilizadas para manipular as informações do banco de dados textual.

5 – Interfaces Amigáveis com o Dialog

De acordo com (JARGAS,2008)**:**

> *O Dialog é um programa de console (modo texto) que desenha janelas na tela, com menus, caixas de texto e botões. Feito especialmente para ser usado com o shell, é excelente para criar interfaces amigáveis para o usuário, fazendo com que ele escolha itens de menu em vez de digitar opções na linha de comando.*

Este utilitário permite criar interfaces amigáveis para programas que são executados no console (modo texto). Este recurso melhora a interação com o usuário, simplificando a operação de programas no shell. Diversas distribuições do Linux utilizam o Dialog para automatizar tarefas de instalações de programas, exibindo somente as informações úteis ao usuário.

O uso do Dialog pode tornar o trabalho do administrador de redes mais produtivo, possibilitando que tarefas diárias e repetitivas sejam simplificadas. Devido a seus recursos, os programas para o console geram códigos menores, eliminando a escrita de código para verificações de validação em muitas situações.

5.1 – Tipos de caixas definidas no Dialog

A Tabela 5.1 apresenta todas as caixas definidas no Dialog. Outras informações podem ser encontradas em [Dickey,2006].

Tabela 5.1 – Tipos de caixas do Dialog

Nome	Descrição do tipo de caixa
calendar	Exibe um calendário para escolha de uma data;
checklist	Exibe uma lista de opções e escolhe várias;
fselect	Digita ou escolhe um arquivo;
gauge	Exibe uma barra de progresso (porcentagem);
infobox	Exibe uma mensagem, sem botões;
inputbox	Digita um texto qualquer;
menu	Exibe um menu e escolhe um item;
msgbox	Exibe uma mensagem e aperta o botão ok;
passwordbox	Digita uma senha;
radiolist	Exibe uma lista de opções e escolha uma;
Tailbox	Exibe a saída do comando tail -f
Tailboxbg	Exibe a saída do comando tail -f (segundo plano).
textbox	Exibe o conteúdo de um arquivo;
timebox	Escolhe um horário;
yesno	Exibe uma pergunta e aperta o botão YES ou No.

5.2 – Parâmetros utilizados no Dialog

O Dialog possui 4 parâmetros obrigatórios que devem ser passados na chamada ao programa. Estes parâmetros são: tipo, texto, altura e largura. A sintaxe básica do Dialog é a seguinte:

dialog --tipo 'texto' altura largura

Exemplo:

dialog –msgbox 'Caixa de mensagem do Dialog' 5 40.

Este exemplo cria uma caixa de mensagem básica utilizando o Dialog. A Figura 5.1 exibe a caixa de mensagem resultante do comando.

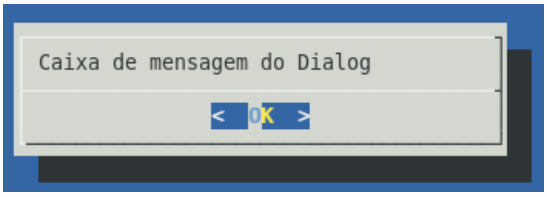

Figura 5.1 – Caixa de mensagem do Dialog

Na Tabela 5.2 são descritas várias opções que podem ser utilizadas para definição dos textos das caixas de mensagens do Dialog.

Tabela 5.2 – Opções para definir textos das caixas

Opção	Descrição
--backtitle texto	Título do topo da tela, que fica no plano de fundo, atrás da caixa.
--title texto	Título da caixa, colocado centralizado na borda superior.
--cancel -label texto	Texto a ser mostrado no botão Cancel.
--exit -label texto	Texto a ser mostrado no botão Exit.
--help -label texto	Texto a ser mostrado no botão Help.
--ok -label texto	Texto a ser mostrado no botao OK.

A tabela 5.3 descreve opções para fazer ajustes no texto das caixas.

Tabela 5.3 – Opções para fazer ajustes no texto das caixas.

Opção	Descrição do comando
--cr-wrap	Mantém as quebras de linhas originais do texto da caixa, para não precisar colocar os \n. Caso a linha fique muito grande, o Dialog irá quebrá-la no meio para caber na caixa.
--no-collapse	Mantém o espaçamento original do texto, além dos TABS e espaços em branco consecutivos
--tab-correct	Converte cada TAB para N espaços. O N é especificado na opção --tab- len ou o padrão 8 é assumido.
--tab- len N	Define o número de espaços que serão colocados no lugar de cada TAB, quando usar a opção –tab-correct.
--trim	Limpa o texto da caixa, apagando espaços em branco no início, espaços consecutivos e quebras de linhas literais.

A Tabela 5.4 descreve opções para fazer vários ajustes nas caixas.

Tabela 5.4 – Opções para fazer ajustes nas caixas

Opção	Descrição
--aspect taxa	Taxa que ajusta o dimensionamento automático das caixas. É a relação largura/altura, sendo o padrão 9, que significa 9 colunas para cada linha.
--begin y x	Define a posição inicial da caixa, relativo ao canto superior esquerdo.
--defaultno	Faz o botão Não ser o padrão da caixa YesNo.
--default-item item	Define qual vai ser o item pré-selecionado do Menu. Por padrão, o primeiro item será selecionado.
--shadow	Desenha a sombra da caixa. Opção já usada normalmente.
--no-shadow	Não desenha a sombra da caixa.

Tabela 5.4 – Opções para fazer ajustes nas caixas (continuação)

Opção	Descrição
--no-cancel	Não mostra o botão Cancel nas caixas Checklist, Inputbox e Menu. A tecla Esc continua valendo para sair da caixa.

A tabela 5.5 descreve opções relativas aos dados informados pelo usuário.

Tabela 5.5 – Opções relativas aos dados informados pelo usuário

Opção	Descrição
--separate-output	Na caixa checklist, retorna os itens selecionados , um por linha e sem aspas.
--separate-widget sep	Define o separador que será colocado entre os retornos de cada caixa. Útil quando se trabalha com múltiplas caixas. O padrão é TAB.
--stderr	Retorna os dados de saída de erros (STDERR). Opção já usada normalmente.
--stdout	Retorna os dados na saída padrão (STDOUT) em vez da STDERR.
--max-input tamanho	Tamanho máximo do texto que o usuário pode digitar nas caixas. O tamanho padrão é 2000 caracteres.

A tabela 5.6 descreve outras opções que podem ser utilizadas no Dialog .

Tabela 5.6 – Outras opções disponíveis no Dialog

Opção	Descrição
--ignore	Ignora as opções inválidas. Serve para manter compatibilidade apenas.
--size-err	Opção antiga que não é mais usada.
--beep	Apita cada vez que a tela é desenhada.
--beep-after	Apita na saída com o Ctrl+C.
--sleep -N	Faz uma pausa de N segundos após processar a caixa. Útil para a Infobox.
--timeout N	Sai do programa com erro caso o usuário não faça nada em N segundos.
--no-kill	Coloca a caixa Tailboxbg em segundo plano(desabilitando seu SIGHUP) e mostra o ID de seu processo na STDERR.
--print-size	Mostra o tamanho de cada caixa na STDERR.
--and-widget	Junta uma ou mais caixas numa mesma tela (sem limpá-la).

A tabela 5.7 descreve outras opções do Dialog que devem ser usadas sozinhas na linha de comando.

Tabela 5.7 – Opções usadas sozinhas na linha de comando

Opção	Descrição
--clear	Restaura a tela caso o dialog a tenha bagunçado.
--create-rc arquivo	Gera um arquivo de configuração do dialog.
--help	Mostra a ajuda do dialog, com as opções disponíveis.
--print-maxsize	Mostra o tamanho atual da tela na STDERR.
--print-version	Mostra a versão do dialog na STDERR.
--version	O mesmo que –print-version.

5.3 – Exemplos de Caixas do Dialog

A seguir serão vistos alguns exemplos de caixas do Dialog. Os comandos estão definidos em mais de uma linha, facilitando a compreensão dos mesmos, mas podem ser escritos em uma mesma linha.

fselect

```
dialog                              \
 --title 'Escolha onde instalar'    \
 --fselect /usr/share/vin/          \
 0 0
```

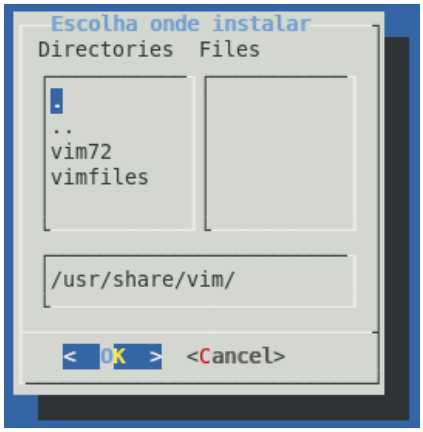

Figura 5.2 – Exemplo de caixa do tipo fselect

inputbox

```
dialog                              \
 --title 'Dados Pessoais'           \
 --inputbox 'Nome completo:'  \
 00
```

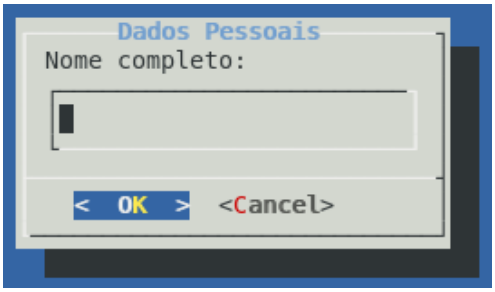

Figura 5.3 – Exemplo de caixa do tipo inputbox

menu

```
dialog                                  \
 --title 'Perfil'                       \
 --menu 'Escolha o perfil de instalação:'  \
 0 0 0                                  \
 mínima    'Instala o mínimo'           \
 completa    'Instala tudo'             \
 customizada 'Você escolhe'             \
```

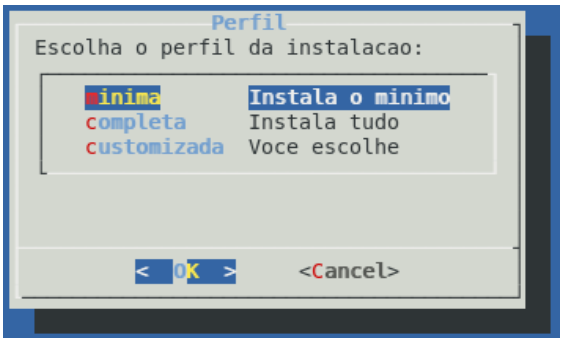

Figura 5.4 – Exemplo de caixa do tipo menu

6 – Implementação da Gerência de Processos

Na introdução deste trabalho foram apresentadas duas hipóteses para o estudo de caso. Após ser feita uma análise das duas opções, optou-se pela segunda opção, que trata do desenvolvimento de um aplicativo em Shell Bash, definido como GPI[14]. Decidiu-se por esta opção pelo motivo de proporcionar um trabalho mais elaborado, com recursos de gerenciamento individual de cada usuário, centralização do controle, além de proporcionar uma melhor interface de comunicação com os usuários. Percebe-se que esta alternativa trará resultados mais eficientes no controle da ociosidade de processos. A Tabela 6.1 apresenta um comparativo entre as duas opções propostas.

Tabela 6.1 – Comparativo de Soluções para Gerenciamento de Processos

Aplicativo Shell GPI	Variável TMOUT
Os processos monitorados poderão ser cadastrados individualmente para cada usuário.	Não haverá cadastro de processos monitorados, pois o que será monitorados será a sessão;
Gerenciamento dos processos por Interface gráfica proporcionada pelo Dialog, simplificando o trabalho do administrador;	Gerenciamento efetuado via Shell, onde os usuários terão seu tempo de inatividade gerenciado manualmente pelo administrador.
Controle de tempo de inatividade individual para cada processo;	Controle de tempo de inatividade por sessão, desta forma todos os processos abertos serão atingidos;
Gerenciamento centralizado por banco de dados	Gerenciamento individual ;

O GPI foi planejado para resolver um problema específico de uma empresa, contudo, da forma como foi implementado, poderá atuar sobre qualquer programa específico em execução no Linux.

14 GPI (Gerenciador de Processos Inativos)

A Figura 6.1 apresenta a tela de processos em execução no sistema sem o monitoramento do GPI.

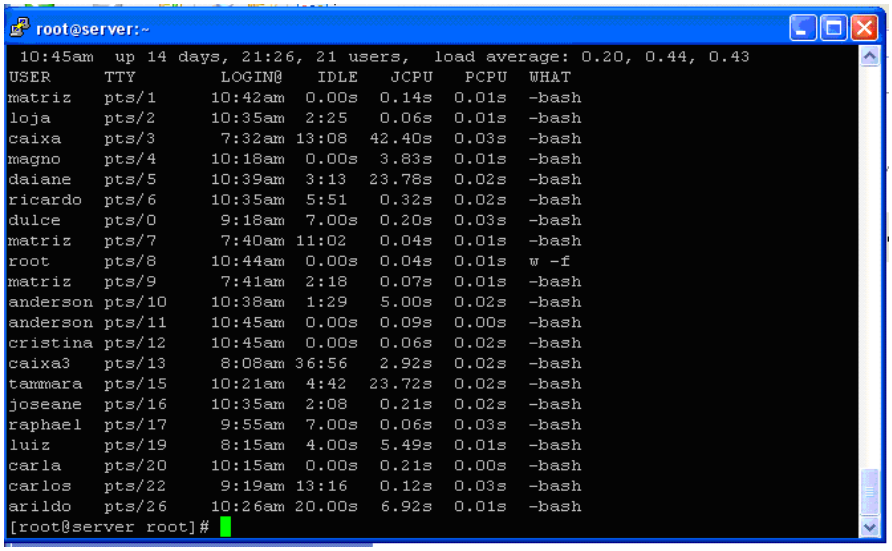

Figura 6.1 – Resultado do comando w -f exibindo processos ociosos no Linux

Através da Figura 6.1 percebe-se que vários processos estão ociosos por diferentes períodos de tempo. O tempo de ociosidade é exibido através do campo *IDLE* da Figura 6.1. A ociosidade dos processos oriundos de programas que possuem controle de licenças de uso prejudica a disponibilidade de programas para os usuários que necessitam utilizá-los. Se o número de acessos limite do programa é atingido, o usuário receberá um aviso de negação de acesso. A Figura 6.2 exibe um exemplo de negação de acesso ao usuário do programa solicitado. Conforme ilustra a figura, todos os usuários que tentarem acessar o sistema neste instante serão negados com a mensagem descrita. Os usuários que tinham o acesso negado eram orientados a contactar o administrador do sistema para solicitar a solução do problema. A tarefa do administrador era usar de procedimentos manuais de verificação de processos ociosos no sistema e eliminar estes processos no servidor.

A tarefa do administrador para liberar os processos ociosos era simples, mas interrompia sua rotina diária de trabalho constantemente, prejudicando a execução de tarefas mais importantes. A falta de acesso ao sistema também demandava tempo dos usuários, que tinham que interromper suas atividades para solicitar ao administrador o acesso ao sistema.

O trabalho manual executado pelo administrador para liberar os processos ociosos é descrito abaixo:

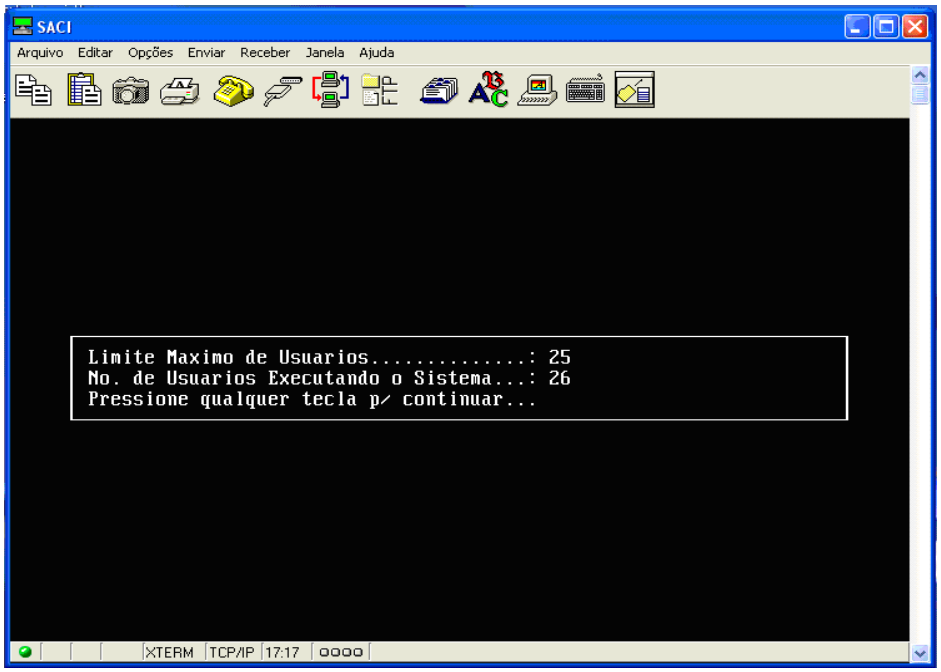

Figura 6.2 – Limite de acessos a sistema atingido

- Digitar no Shell do Linux o comando *w -f*, que exibe na tela os

processos ociosos, conforme é exibido na Figura 6.1;

- Identificar na Figura 6.1 quais os terminais de usuários com o tempo de ociosidade maior.
- Digitar no Shell do Linux o comando *ps -aux | grep terminal,* onde *terminal* é a máquina do usuário identificada com ociosidade maior na Figura 6.1;
- Localizar o [15]PID do processo ocioso alvo do usuário para ser terminado;
- Digitar no Shell do Linux o comando kill PID;

Com o procedimento adotado pelo administrador, citado acima, o processo selecionado foi finalizado. O próximo usuário que tentar conectar ao sistema obterá o acesso, pois ocupará o espaço do processo finalizado.

O GPI foi implementado com uma interface gráfica que permita ao administrador de redes cadastrar todos os usuários que pretenda monitorar. Cada cadastro será vinculado a um programa que necessite do controle de sua ociosidade, entretanto, um mesmo usuário poderá ter vários processos monitorados. Os dados cadastrados serão armazenados em um banco de dados textual, que será manipulado através de procedimentos implementados no aplicativo.

Após a definição do cadastro dos usuários e programas monitorados, uma rotina externa denominada */src/monitora.sh* fará o trabalho de monitoramento dos processos ociosos. A figura 6.3 exibe o GPI instalado no servidor, através do qual fará o monitoramento das estações com processos ociosos.

15 Número no sistema Linux que identifica um processo em execução

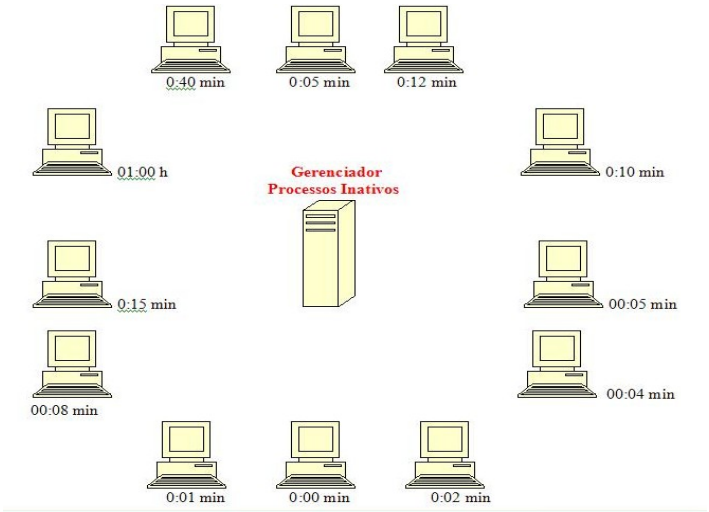

Figura 6.3 – GPI no servidor com estações a serem monitoradas

6.1 – Scripts e Arquivos da Aplicação

Para o perfeito funcionamento do GPI é necessário ter instalados no sistema todos os scripts e arquivos que pertençam à aplicação. Na Tabela 6.2 são listados os scripts e arquivos que integram o GPI.

Tabela 6.2 – Scripts e Arquivos do Aplicativo GPI

Nome do arquivo	Descrição do arquivo
consulta.awk	Arquivo com comandos awk que formatam a saída de consultas de processos monitorados;
gerbdtext.sh	Script que gerencia o banco de dados textual, com funções de inclusão, alteração, exclusão e consultas;
gpi.sh	Script do aplicativo que permite o cadastro de usuários e programas monitorados;

Tabela 6.2 – Scripts e Arquivos do Aplicativo GPI (continuação)

monitora.sh	Script responsável pelo monitoramento de programas cadastrados no banco de dados textual;
process.txt	Arquivo que armazena o banco de dados textual do aplicativo;

Entre os scripts e arquivos do GPI, o que tem mais importância na execução do monitoramento é o *monitora.sh*, pois é através dele que ocorre o monitoramento de ociosidade dos processos cadastrados.

6.1.1 – Arquivo consulta.awk

O arquivo consulta.awk é composto de comandos awk[16], que formatam a saída de arquivos texto. No GPI ele é utilizado para formatar a saída dos processos cadastrados.

A Figura 6.4 exibe o código utilizado no arquivo consulta.awk.

```
1  BEGIN { FS = ","
2        printf "\n%-12s %-20.20s %-12.12s %-8.8s\n","LOGIN", "NOME USUARIO","PROCESSO","TEMPO"
3        print "-----------------------------------------------------"}
4     { printf "%-12.12s %-20.20s %-12.12s %-8.8s\n",$1, $3, $2, $4}
5  END  {print "-----------------------------------------------------"}
```

Figura 6.4 – Arquivo consulta.awk

Os comandos print e printf utilizados nas linhas 2 a 5 da Figura 6.4 definem a formatação da saída dos dados de acordo com o cabeçalho na linha 2. Pode-se perceber que o comando printf utiliza vários argumentos para definir a formatação dos dados. Maiores detalhes sobre o comando printf podem ser encontrados em (LAUREANO,2005).

16 O Awk é considerado uma linguagem de programação direcionada a processamento de textos.

6.1.2 – Arquivo gerbdtext.sh

O arquivo gerbdtext.sh é um script que exerce a função de gerenciador do banco de dados. Neste arquivo estão codificadas funções que executam operações básicas de manutenção do banco de dados. Entre estas operações estão: inclusão, alteração, exclusão, consulta e validação de informações manipuladas. As funções do script gerbdtext.sh serão chamadas pelo GPI.

A Figura 6.5 exibe a função de inclusão de registros no banco de dados textual.

```
# insere o registro $* no banco
insere_registro() {
  local chave=$(echo "$1" | cut -d $SEP -f1)  # pega primeiro campo
  echo "$*" >>"$BANCO"                        # grava o registro
  echo "Registro de '$chave' cadastrado com sucesso."
  return 0
}
```

Figura 6.5 – Inclusão de registros no banco de dados

6.1.3 – Script gpi.sh

No arquivo gpi.sh estão codificados os comandos que criam a interface do sistema. Todas as operações de gerenciamento dos processos são cadastradas através da interação do usuário com esta interface. As operações são executadas através de opções de menus criados com o Dialog. O gpi.sh interage com o gerbdtext.sh através de chamadas às funções que acessam o banco de dados textual. Para o perfeito funcionamento do GPI são definidos no gpi.sh alguns parâmetros necessários para as operações do sistema. Os parâmetros definidos no gpi.sh são:

- Definição da variável BANCO, que receberá o nome do arquivo textual que representará o banco de dados;
- Definição da variável CRONTAB, que receberá o endereço da localização do arquivo crontab;
- Utilização do comando source[17] para incluir no gpi.sh as funções de manipulação do banco de dados definidas em gerbdtext.sh;

6.1.4 – Script monitora.sh

O script monitora.sh é responsável pelo processamento dos dados cadastrados no banco de dados do GPI. Ao ser executado, o monitora.sh processa todos os registros em busca de processos em execução no Linux, que estão ociosos além do tempo definido nos registros. A frequência de execução do monitora.sh é definido no GPI através de configurações definidas no arquivo crontab[18] do Linux. As ações executadas pelo script monitora.sh são exibidas na Tabela 6.3.

Tabela 6.3 – Algoritmo do monitora.sh

Passo	Descrição
1	Inicia leitura do banco de dados textual do GPI (process.txt)
2	Armazena nas variáveis login, processo, tempo e código, os dados do registro atual, correspondente ao login do usuário, processo cadastrado, tempo máximo de inatividade e ao UID do usuário cadastrado, respectivamente.
3	Executa teste no registro, para verificar se o registro atual não é o registro de cabeçalho do banco de dados (definido no primeiro registro).

17 O comando source seguido do nome de um script, deixa disponíveis no script que está definido, todas as funções definidas no script chamado.

18 É um programa do Linux que edita o arquivo onde são especificados os comandos a serem executados e a hora e dia de execução pelo cron (um programa que executa tarefas agendadas no Linux)

Tabela 6.3 – Algoritmo do monitora.sh (continuação)

4	Verifica o tamanho da variável login acrescentando espaços até 8 caracteres quando se tratar de login com tamanho menor que 8 caracteres.
5	Efetua uma pesquisa com o comando *w,* juntamente com o *grep ,* em busca do login do registro atual, direcionando o resultado para um arquivo temporário. Com a pesquisa, todos os processos do login serão direcionados para o arquivo temporário. Exemplo: dados=$(w -f \| grep -i "$login" | grep -i bash >arqtemp)
6	Inicia leitura no arquivo temporário, enquanto existir registros do login.
7	Armazena nas variáveis *tempo_inativo* e *terminal ,* o tempo que de inatividade do processo atual e o terminal que está executando o processo, respectivamente.
8	Define o valor de *tempo_inativo* como zero, caso o seu valor seja menor do que um minuto.
9	Armazena nas variáveis *hr1, mm1, hr2, mm2*, as horas e minutos da variáveis *tempo* do passo 2 e *tempo_inativo* do passo 7, respectivamente.
10	Caso a variável *hr2* não tenha valor válido em horas, atribui seu valor igual a zero.
11	Converte o conteúdo das variáveis *hr1* e *hr2* para minutos adicionando à seus valores *mm1 e mm2*, respectivamente. Após a soma, armazenar o resultado nas variáveis *n1* e *n2*.
12	Se o tamanho do *login* do usuário for maior de 8 caracteres, então assume que o valor do login = codigo do passo 2.
13	Armazena na variável *n0* a diferença entre *n1* e *n2*.
14	Se *n0* < 5 e *n2* >0 então localiza o *PID* do processo atual como segue: PID=$(ps aux \|grep -i "$login" | grep -i "$processo" \| grep -i "$terminal" |tr -s ' ' |cut -d ' ' -f 2). Obs: Esta condição é acionada quando faltar menos de 5 minutos para atingir o tempo limite.
15	Coloca o processo em pausa para emitir aviso de alerta para o usuário. Ex: kill -19 $PID
16	Emite o aviso para o *terminal* como segue: SISTEMA SEM USO !!! PODERÁ SER FECHADO EM INSTANTES.
17	Após emitir o aviso, volta o estado do processo ao estado normal. Ex:kill -18 $PID

Tabela 6.3 – Algoritmo do monitora.sh (continuação)

18	As variáveis *n1* e *n2* citadas no passo 11, armazenam o tempo limite do processo no cadastro e o tempo atual de inatividade, respectivamente. Quanto o valor de *n2* se torna maior do que *n1,* o processo deverá ser fechado, conforme segue abaixo: localiza o processo: PID=$(ps aux \|grep -i "$login" | grep -i "$processo" \| grep -i "$terminal" |tr -s ' ' |cut -d ' ' -f 2) fecha o processo: kill "$PID".
19	Após o processamento completo do arquivo temporário do passo 5, volta o controle para o arquivo do banco de dados (process.txt) do passo 1. Seguindo com o processamento até o último registro.

A Figura 6.6 exibe a tela de um processo sendo monitorado pelo script monitora.sh.

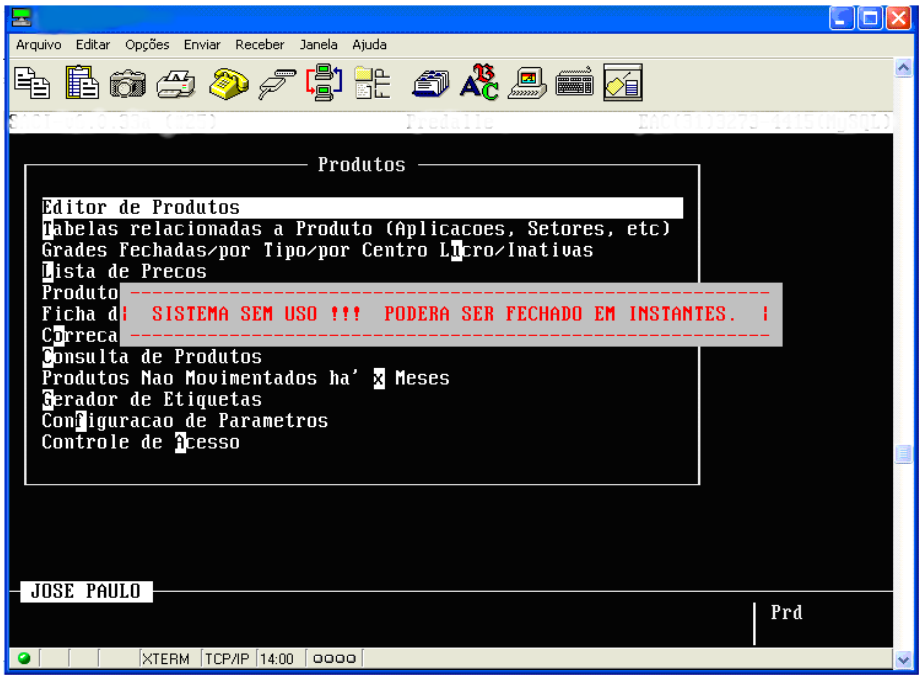

Figura 6.6 – Mensagem ao usuário acionada pelo script monitora.sh

A figura 6.7 exibe três estações com mensagem de alerta conforme a figura 6.6. A mensagem é enviada cinco minutos antes do vencimento do tempo limite de inatividade do processo monitorado.

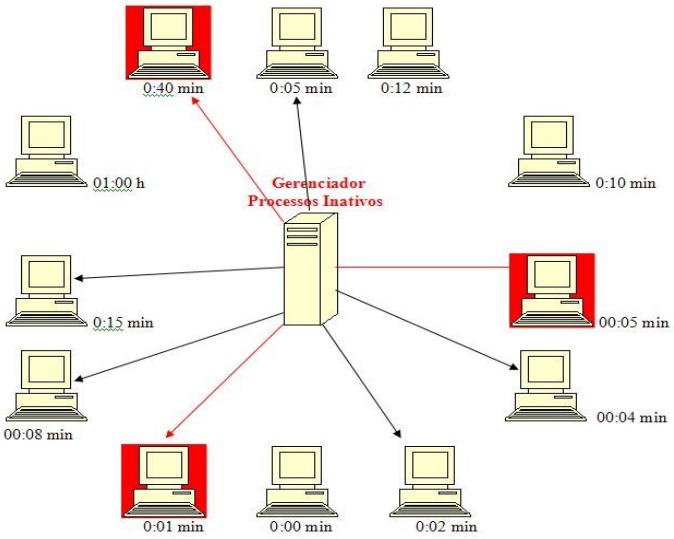

Figura 6.7 – Envio de mensagens de alerta para os processos monitorados

6.1.5 – Arquivo process.txt

O arquivo process.txt armazena o banco de dados textual utilizado no GPI. Após as atualizações no GPI, as informações são armazenadas neste arquivo. A estrutura do arquivo process.txt é descrita abaixo:

login,processo,nome_usuario,tempo_inatividade

A Tabela 6.4 exibe a descrição dos campos do arquivo process.txt.

Tabela 6.4 – Descrição dos campos do arquivo process.txt

Campo	Descrição
login	Login do usuário no Linux
processo	Nome do processo a ser monitorado pelo GPI
nome_usuario	Nome completo do usuário que terá seu processo monitorado
tempo_inatividade	Tempo máximo permitido para que o processo fique inativo.

A Figura 6.8 exibe o arquivo process.txt com vários processos cadastrados. Pode-se perceber que a estrutura do arquivo process.txt utiliza a vírgula como delimitador entre os campos.

```
login,processo,nome_usuario,tempo_inatividade
jose,saci,Jose Paulo da Silva,00:05
adriano,saci,Adriano,00:30
aniceto,saci,Aniceto,00:30
carlos,saci,Carlos,00:30
rocha,saci,Rocha,00:30
luiz,saci,Luiz,00:30
roberto,saci,Roberto,00:30
```

Figura 6.8 – Exemplo de registros do arquivo process.txt

6.2 – Menu Principal da Aplicação

Ao iniciar o aplicativo é apresentado um menu com as opções das operações básicas do sistema.

A Figura 6.9 apresenta a tela do menu principal do GPI.

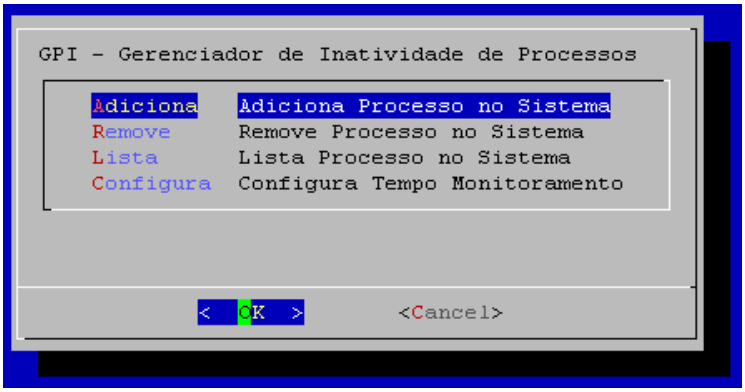

Figura 6.9 – Tela Principal do Aplicativo GPI

6. 3 – Adicionar Processos

Ao adicionar processos no monitoramento, o GPI automaticamente faz a leitura de todos os usuários do Linux, evitando erros de digitação, bastando apenas o usuário escolher qual usuário será monitorado. A Figura 6.10 mostra a tela de escolha do login a incluir.

Após a tela apresentada na Figura 6.10 será solicitada a digitação do nome do usuário conforme a Figura 6.11.

Em seguida, o sistema irá solicitar o nome do processo que deverá ser monitorado. O processo ficará vinculado ao login do usuário informado na Figura 6.10.

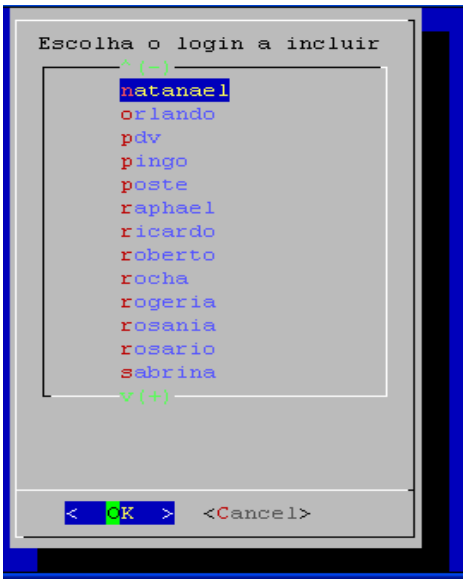

Figura 6.10 – Adicionar processos

O GPI permite que vários processos sejam cadastrados para o mesmo usuário, bastando informar nas próximas inclusões o mesmo nome de login e o nome do novo processo a ser monitorado. A Figura 6.12 exibe a tela de solicitação do processo.

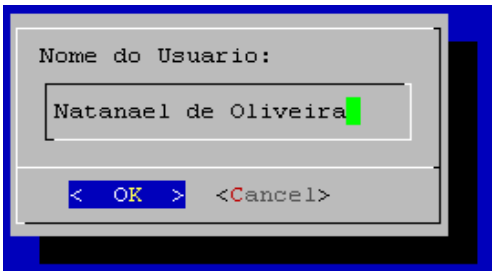

Figura 6.11 – Entrada do nome do usuário

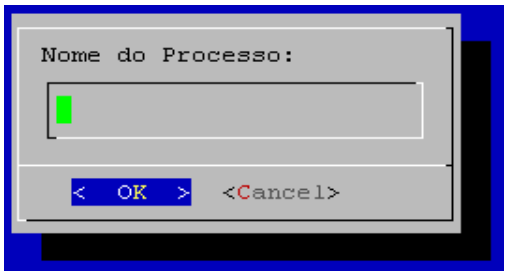

Figura 6.12 – Entrada do nome do processo

Após a digitação do nome do processo, o GPI executará uma pesquisa no sistema para verificar se o processo informado existe. Se não existir será retornado uma mensagem de erro. Em seguida, o usuário deverá informar o tempo de ociosidade permitido para o processo informado. A Figura 6.13 exibe a tela de entrada do tempo de inatividade. O tempo informado deverá estar de acordo com o formato especificado na tela de entrada.

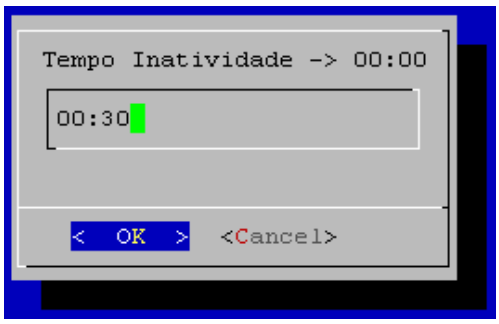

Figura 6.13 – Tempo de inatividade do processo

Se todas as informações solicitadas foram preenchidas corretamente, o sistema emitirá uma aviso, confirmando a inclusão do registro no banco de dados. A Figura 6.14 demonstra este resultado.

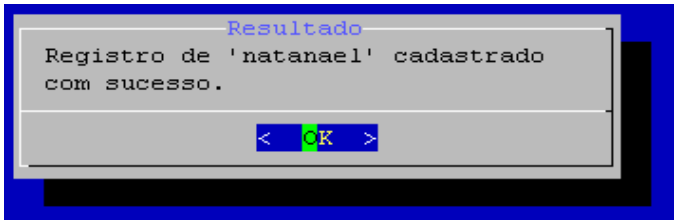

Figura 6.14 – Aviso de inclusão de registro

6.4 – Remover Processos

A opção *Remove* do menu principal retorna para o usuário todos os processos cadastrados, isto facilita a escolha de qual processo a remover, evitando erros de digitação. Esta opção exibe os usuários e o processos cadastrados, o que pode ser observado na Figura 6.15.

6.5 – Listar Processos

A opção *Lista* do Menu Principal possibilita que sejam exibidos todos os processos cadastrados no GPI. A Figura 6.17 exibe a tela de exemplo com vários processos cadastrados. Nesta figura são listados o login do usuário, o nome do usuário, o processo monitorado e o tempo máximo de ociosidade do processo.

6.6 – Configurar Tempo de Monitoramento

A opção *configura* do menu principal é utilizada para determinar como o [19]crontab irá executar o código de monitoramento dos processos. No

19 É um programa do Linux que edita o arquivo onde são especificados os comandos a serem executados e a hora e dia de execução pelo cron (um programa que executa

crontab será configurado qual a frequência que o GPI será executado, que verifica no sistema quais os processos cadastrados estão conforme o especificado no tempo limite de inatividade. A Figura 6.16 exibe a tela de configuração do monitoramento.

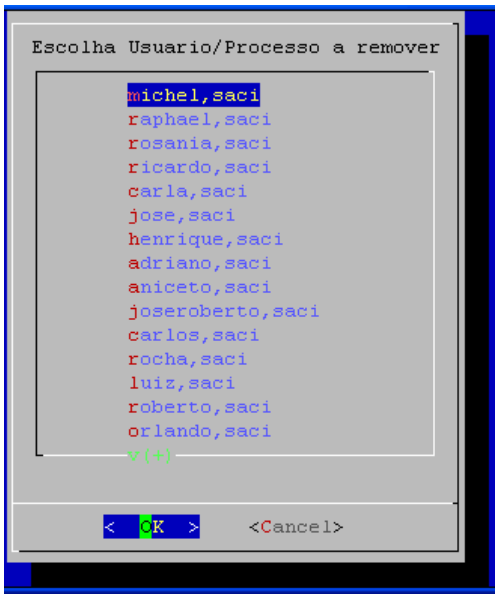

Figura 6.15 – Remover processos

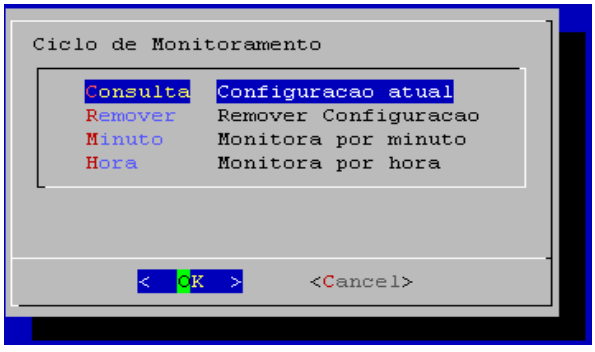

Figura 6.16 – Configuração de tempo de monitoramento

tarefas agendadas no Linux)

Figura 6.17 – Lista de processos monitorados

6.6.1 – Consultar Configuração

A opção *Consulta* da Figura 6.16 exibe a configuração da frequência de monitoramento. A Figura 6.18 exibe a configuração atual.

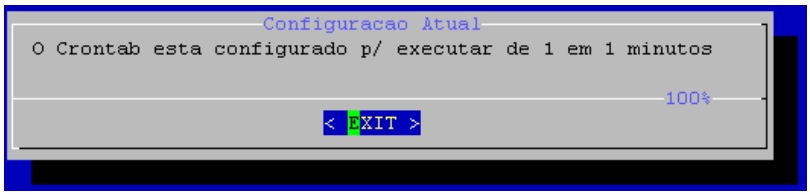

Figura 6.18 – Consulta configuração atual

6.6.2 – Remover Configuração

A opção *remover* da Figura 6.16 permite remover a configuração atual do tempo de monitoramento. Ao removê-la o monitoramento será suspenso até que uma nova configuração seja efetuada. Esta opção não irá remover os processos cadastrados, mas somente a linha configurada no crontab que indica a frequência de execução do GPI no sistema operacional.

6.6.3 – Monitoramento Por Minuto

A opção *Minuto* da Figura 6.16 permite que seja definido uma frequência de execução do monitoramento em minutos. A Figura 6.19 exibe a tela de entrada da frequência de minutos a monitorar.

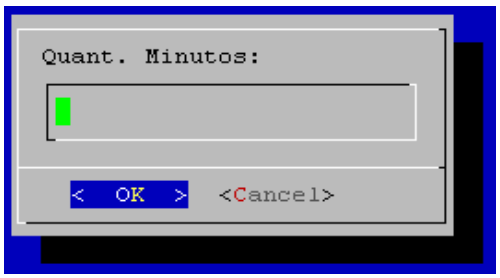

Figura 6.19 – Monitoramento por minuto

6.6.4 – Monitoramento Por Hora

A opção *Hora* da Figura 6.16 permite que seja definida uma frequência de execução do monitoramento por hora. A Figura 6.20 exibe a tela de entrada da frequência de horas a monitorar.

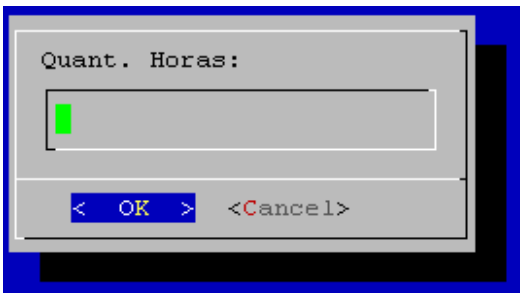

Figura 6.20 – Monitoramento por hora

7 – Testes Realizados e Resultados

Foi criado um ambiente para execução de testes e correções, utilizando a distribuição Red Hat Linux release 7.3 e Dialog 1.1-20080819.

Nos testes realizados o aplicativo se mostrou bastante funcional, oferecendo acesso fácil as opções, com interfaces limpas e amigáveis para o administrador.

De acordo com os resultados, este trabalho atingiu o objetivo para qual foi proposto. A indisponibilidade do sistema foi solucionada, pois não foi detectado nenhum problema de acesso após o funcionamento do aplicativo. Os benefícios refletiram principalmente no setor de vendas, melhorando as condições de trabalho no atendimento aos clientes. No setor administrativo eliminou-se a espera por acesso ao sistema quando havia necessidade de uso, evitando a ociosidade dos funcionários. Além dos benefícios para os usuários, houve melhora no desempenho das atividades do administrador de redes, que não precisou mais interromper suas atividades para atender a usuários com problemas na disponibilidade de acesso.

Os resultados refletem também na parte econômica da empresa, pois com o gerenciamento eficiente das licenças do software proprietário haverá uma redução no investimento em aquisições de novas licenças, gerindo os recursos computacionais com mais eficácia. O administrador passou a realizar o monitoramento dos processos de uma forma prática, simples e automatizada.

As intervenções do administrador são necessárias somente quando houver um aumento de estações ou inclusão de novos usuários, cadastrando-os no monitoramento ou diminuindo o tempo de ociosidade permitido para os usuários cadastrados.

8 – Conclusão

O objetivo principal deste trabalho foi desenvolver uma aplicação que controlasse o uso de um aplicativo específico utilizado por uma empresa, no entanto, na codificação dos scripts foi adicionado um importante recurso, que é a sua abrangência para outros aplicativos no ambiente Linux. Desta forma, basta definir o aplicativo a ser monitorado e cadastrá-lo para que seu monitoramento seja feito.

Criar soluções que visam a redução de custos com aquisição de licenças de software proprietário e automatização de tarefas do administrador são fatores que tornam o departamento de informática de uma empresa mais eficiente e eficaz.

O GPI conseguiu atender de forma satisfatória o seu objetivo proposto, controlando os processos inativos de forma eficiente, eliminando custos adicionais com licenças de software, além de tornar o trabalho do administrador mais produtivo.

9 – Referências Bibliográficas

JARGAS, Aurélio Marinho. **Expressões Regulares:** Uma abordagem divertida. São Paulo: Novatec Editora, 2006.

JARGAS, Aurélio Marinho. **Shell Script Profissional.**São Paulo: Novatec Editora, 2008.

CAMARGO, Herlon Ayres. **Automação de Tarefas.** Lavras: UFLA/FAETE, 2005.

TANENBAUM, A.S and WOODHULL. A. S. **Operating systems**: Design and Implementation. Prentice Hall, New Jersey - USA, 2nd edition,1997

DICKEY, T.. Manpage of DIALOG (Updates for 0.9b). [on-line]. Disponível na Internet via www. url: http://hightek.org/dialog/manual-0.9a-20010429.html#index.
Arquivo capturado em 28 de dezembro de 2008.

NEVES, J. C. **Programação Shell Linux**. Rio de Janeiro: Brasport, 2005. 408p.

BURTCH, K. O. **Scripts de Shell Linux com Bash**. Rio de Janeiro: Ciência Modera,2005. 522p.

Apêndice A

Código fonte do arquivo gpi.sh

```
#!/bin/bash
#
# process_conf.sh - Configura processos para monitoramento de inatividade de
# uso no sistema
#
# Requisitos: gerbdtext.sh, dialog
#
# 2009-10-07 José Paulo da Silva
#
#

#-------------------------[ Configuração ]---------------------------------

# Localização do arquivo do banco de dados textual
BANCO=process.txt

# Localizacao do arquivo crontab
CRONTAB=/etc/crontab

# Inclui o gerenciador do banco textual
source gerbdtext.sh || {
  echo "Ops, ocorreu algum erro no gerenciador do banco"
  exit 1
}

while :
do
acao=$(dialog --stdout \
   --menu "Monitoramento de Inatividade de Processos" \
   0 0 0 \
   Adiciona  "Adiciona Processo no Sistema" \
   Remove    "Remove Processo no Sistema"\
   Lista     "Lista Processo no Sistema"\
   Configura "Configura Tempo Monitoramento")

   [ $? -ne 0 ] && exit

# Tratando os comandos recebidos

case "$acao" in

   Adiciona)
     # obtem lista de logins
     usuarios=$(awk -F ":" '$3>499 {print $1}' /etc/passwd |sed 1d |sort>arq)
     # le arquivo temporário
     item_menu=""
     while read linha
```

```
do
  item_menu=$(echo $item_menu$linha" """""" ")
done < arq
usuarios=$(echo "$item_menu");
login=$(eval dialog --stdout \
   --menu \"Escolha o login a incluir\" \
   0 0 0 $usuarios)
[ $? -ne 0 ] && continue
nome_usuario=$(dialog --stdout --inputbox "Nome do Usuario:" 0 0)
[ $? -ne 0 ] && continue
[ -z $nome_usuario ] && {
   msg="Nome usuario em branco!!!"
   dialog --msgbox "$msg" 6 40
   continue
}
processo=$(dialog --stdout --inputbox "Nome do Processo:" 0 0)
[ $? -ne 0 ] && continue
[ -z "$processo" ] && {
   msg="Nome Processo em branco !!!"
   dialog --msgbox "$msg" 6 40
   continue
}

# verifica se programa existe no servidor
ok="nao"
find / -name "$processo" >proc_arq
while read proc
do
 arq=$(echo "$proc")
 [ -x "$arq" ] && ok="sim"
done  < proc_arq


[ $ok = "nao" ] && {
  msg="Processo nao existe !!!"
  dialog --msgbox "$msg" 6 40
continue
}

# verifica se já existe o login e o processo cadastrado
  tem_chave_2 "$login","$processo" && {
  msg="O Login e Processo já foi cadastrado"
  dialog --msgbox "$msg" 6 40
  continue
  }

tempo_inatividade=$(dialog --stdout --inputbox "Tempo Inatividade -> 00:00"  0 0)
[ $? -ne 0 ] && continue

# validando hora informada
valida_hora=$(echo "$tempo_inatividade" | grep [012][0-9]:[0-5][0-9])
[ -z $valida_hora ] && {
   msg="Tempo informado invalido !!!"
   dialog --msgbox "$msg" 6 40
   continue
 }
```

```
  # pronto para inserir os dados
  msg=$(insere_registro "$login,$processo,$nome_usuario,$tempo_inatividade")
  dialog --title "Resultado" --msgbox "$msg" 6 40
 ;;

Remove)
  # obtem Lista de processos gerando arquivo temporario
  login=$(localiza_campo 1,2 | sed 1d)
  login=$(echo "$login" >arq)
  # le arquivo temporario
  item_menu=""
  while read linha
  do
   item_menu=$(echo $item_menu$linha" """""" ")
  done <arq
  processos=$(echo "$item_menu")
  login=$(eval dialog --stdout \
    --menu \"Escolha Usuario/Processo a remover\" \
    0 38 0 $processos)
  [ $? -ne 0 ] && continue

  login=$(echo "$login" |sed 's/---/,/')
  msg=$(apaga_registro "$login")
  dialog --title "Resultado" --msgbox "$msg" 6 40
 ;;

Lista)
    # Lista dos processos monitorados (apaga a primeira linha)
    cat process.txt |sed 1d >arq
    awk -F"," -f consulta.awk arq >arq1
    dialog --title "Processos Monitorados" --textbox arq1 20 59
    rm arq1
 ;;

Configura)
  while :
  do
   # Configura o tempo de monitoramento de processos
   acaoConf=$(dialog --stdout --menu \
   "Ciclo de Monitoramento" 0 0 0 \
   Consulta "Configuracao atual" \
   Remover  "Remover Configuracao" \
   Minuto "Monitora por minuto" \
   Hora "Monitora por hora")
   [ $? -ne 0 ] && break
   conf_ok="0"
   case "$acaoConf" in
       Minuto)
        # Configura monitoramento por minuto
          quant_minuto=$(dialog --stdout --inputbox "Quant. Minutos:" 0 0)
          [ $? -ne 0 ] && continue
          # inserir configuração, se existir atualiza
          linha_config='*/'$quant_minuto' * * * * root /home/monitora.sh'
          existe_crontab 'monitora.sh' && {
```

```
    msg="Configuracao ja existente !!!"
    dialog --title "Resultado" --msgbox "$msg" 6 40
    continue
   }
   # validando quantidade de minutos informada
   valida_minuto=$(echo "$quant_minuto" |grep \
   "^[1-5]\{1\}\?[0-9]\{1\}\?$")
   [ -z "$valida_minuto" ] && {
        msg="Quant. Invalida, informe entre 1-59 !!!"
        dialog --msgbox "$msg" 6 45
        continue
   } || conf_ok="1"
;;
Hora)
 # Configura monitoramento por hora
   quant_hora=$(dialog --stdout --inputbox "Quant. Horas: " 0 0)
   [ $? -ne 0 ] && continue
   # inserir configuração, se existir atualiza
   linha_config='* */'$quant_hora' * * * root /home/monitora.sh'
   existe_crontab "monitora.sh" && {
    msg="Configuracao ja existente !!!"
    dialog --title "Resultado" --msgbox "$msg" 6 40
    continue
   }
   # validando quantidade de horas informada
   valida_hora=$(echo "$quant_hora" |grep \
   "^\(1[0-9]\{1\}\|2[0-4]\{1\}\|[1-9]\{1\}\)$")
  [ -z "$valida_hora" ] && {
       msg="Quant. Invalida, informe entre 1-24 !!!"
       dialog --msgbox "$msg" 6 45
      continue
   } || conf_ok="1"
;;
Consulta)
 # consulta configuração atual no Crontab
 # verifica se existe uma configuração (minuto ou hora)
   existe_crontab "monitora.sh" || {
   msg="Nao existe nenhuma configuracao !!!"
  dialog --title "Resultado" --msgbox "$msg" 6 40
  continue
  }
  # verifica se existe configuracao em horas ou minutos
     # verifica se a configuração é de minutos ou horas
       pesq1=$(grep -i "^*/[1-5]\{1\}\?[0-9]\{1\}.*monitora.sh$" "$CRONTAB")
       pesq1=$(echo "$pesq1" |cut -d " " -f1 |cut -c3-)
       pesq2=$(grep -i "^\* \*/\(1[0-9]\{1\}\|2[0-4]\{1\}\|[1-9]\{1\}\).*monitora.sh$" "$CRONTAB")
       pesq2=$(echo "$pesq2" |cut -d " " -f2 |cut -c3-)
       [ -n "$pesq1" ] && mens="O Crontab esta configurado p/ executar de $pesq1 em $pesq1 minutos"
       [ -n "$pesq2" ] && mens="O Crontab esta configurado p/ executar de $pesq2 em $pesq2 horas"
       echo "$mens" >arq3
       dialog --title "Configuracao Atual" --textbox arq3 6 63
       rm arq3
 ;;
Remover)
 # Consulta configuração atual no Crontab
 # Verifica se existe uma configuração (minuto ou hora)
```

```
            existe_crontab "monitora.sh" || {
            msg="Nao existe nenhuma configuracao !!!"
            dialog --title "Resultado" --msgbox "$msg" 6 40
            continue
            }
            # Localiza a Configuração para remover
              pesq1=$(grep -i "^*/[1-5]\{1\}\?[0-9]\{1\}.*monitora.sh$" "$CRONTAB")
              pesq2=$(grep -i "^\* \*/\(1[0-9]\{1\}\|2[0-4]\{1\}\|[1-9]\{1\}\).*monitora.sh$" "$CRONTAB")
              [ -n "$pesq1" ] && { grep -i -v "^*/[1-5]\{1\}\?[0-9]\{1\}.*monitora.sh$"  "$CRONTAB" >
"$TEMP"
                mv "$TEMP" "$CRONTAB"
                msg="A configuracao foi removida"
              }
              [ -n "$pesq2" ] && { grep -i -v "^\* \*/\(1[0-9]\{1\}\|2[0-4]\{1\}\|[1-9]\{1\}\).*monitora.sh$"
"$CRONTAB" > "$TEMP"
                mv "$TEMP" "$CRONTAB"
                msg="A configuracao foi removida"
              }
              dialog --title "Resultado" --msgbox "$msg" 6 40
          ;;
      esac
        # Grava no arquivo crontab
          [ "$conf_ok" = "1" ] && {
            msg=$(insere_registro_crontab "$linha_config")
            dialog --title "Resultado" --msgbox "$msg" 6 40
          }
 done

esac

done
```

Apêndice B

Código fonte do arquivo monitora.sh

```
#!/bin/bash
#
# monitora.sh - Monitora processos sem atividade em tempo configurável
#
#
# 2009-10-07 José Paulo da Silva
#
#
#


# Faz varredura no arquivo process.txt em busca de processos inativos
while read linha
do

login=$(echo "$linha" | cut -d , -f 1)
processo=$(echo "$linha" | cut -d , -f 2)
tempo=$(echo "$linha" | cut -d , -f 4)
codigo=$(echo "$linha" | cut -d , -f 5)


if test "$login" != "login" ; then

 # verifica tamanho da variavel login, aumenta espacos ate o limite para evitar
 # pesquisar mais de um nome diferente no grep

   tam=$(echo $login |wc -L)
   df=$((8-$tam))
   [ $df -gt 0 ] && {
     for (( i=1; $i <= $df; i++))
     do
       login=$login' '
     done
   }
   # define o login com o maximo de 8 caracteres
     login=$(echo "$login" |cut -c 1-8)

 # pesquisa pelo comando w o usuario selecionado criando um arquivo temporario
  dados=$(w -f | grep -i "$login" | grep -i bash >arqtemp)

  ## faz leitura no arquivo temporário
  while read linha_arqtemp
  do

    tempo_inativo=$(echo "$linha_arqtemp" |tr -s ' ' |cut -d ' '  -f 4)
    terminal=$(echo "$linha_arqtemp" |tr -s ' ' |cut -d ' ' -f 2)

    # zera tempo quando se tratar de segundos
```

```
  hora_seg=$(echo "$tempo_inativo" |grep -i "s")
  [ "$hora_seg" ] && tempo_inativo="0"

  # extrai horas e minutos para variáveis separadas
  hr1=${tempo%:*}
  mm1=${tempo#*:}
  hr2=${tempo_inativo%:*}
  mm2=${tempo_inativo#*:}

  testahr2=$(echo "$hr2" | grep -i 'm')
  # verifica se campo de horas está em minutos
  if test -z "$testahr2"
  then
    mm2="$hr2"
    hr2="0"
  fi

  # retira o s ou m do final do campo de horas (segundos e minutos)
  mm2=$(echo "$mm2" |tr -d 'm' |tr -d 's')

  # Retira o zero das horas e minutos menores que 10
  hr1=${hr1#0}
  mm1=${mm1#0}


  hr2=${hr2#0}

  mm2=${mm2#0}

  # Converte o tempo todo para minutos
  n1=$((hr1*60+mm1))
  n2=$((hr2*60+mm2))

  # Se o tamanho do login original no /etc/passwd for maior de 8 caracteres,
  # assume o login = numero do codigo do usuario
    [ $tam -gt 8 ] && login="$codigo"



  # Se o tempo inativo for menor ou igual a 5 minutos, começa a emitir aviso para o usuario
  n0=$(($n1-$n2)) # guarda a diferenca entre os campos de tempo


  [ $n0 -le 5 -a $n2 -gt 0 ] && {
    # localiza o PID
      PID=$(ps aux |grep -i "$login" | grep -i "$processo" | grep -i "$terminal" |tr -s ' ' |cut -d ' ' -f 2)
    # coloca o processo em pausa para emitir o aviso
      kill -19 $PID
    # emitindo o aviso para o terminal

     echo -e '\033[47;31;1;5m\033[9;11H ---------------------------------------------------------- \033[m' >/dev/
$terminal

     echo -e '\033[47;31;1;5m\033[10;11H|  SISTEMA SEM USO !!!  PODERA SER FECHADO EM
INSTANTES.  | \033[m'>/dev/$terminal
```

```
     echo -e '\033[47;31;1;5m\033[11;11H                                                \033[m'>/dev/$terminal

     echo -e '\033[47;31;1;5m\033[11;11H ------------------------------------------------------------ \033[m'>/dev/
$terminal

     echo -e '\033[47;31;1;5m\033[9;11H                                                 \033[m'>/dev/$terminal

     echo -e '\033[47;31;1;5m\033[9;11H ------------------------------------------------------------ \033[m'>/dev/
$terminal

     # apos o aviso volta o processo ao estado anterior
       kill -18 $PID
    }


    if test  $n2 -gt $n1
    then
      # Localiza o PID do processo
       PID=$(ps aux |grep -i "$login" | grep -i "$processo" | grep -i "$terminal" |tr -s ' ' |cut -d ' ' -f 2)
       kill "$PID"

   fi

  done < arqtemp

fi
done < /opt/gpi/process.txt
```

Apêndice C

Código fonte do arquivo gerbdtext.sh

```
# gerbdtext.sh - Gerenciador de Banco Textual
#
#
# Biblioteca de funções para gerenciar os dados do banco textual
# para utilizá-la é preciso chama-la como o comando source no shell
# Exemplo: source gerbdtext.sh
#
# 2009-10-07 José Paulo da Silva
#

#--------------------------[ Configuração ]--------------------------------

SEP=,            # defina aqui o separador, padrao é :
TEMP=temp.$$     # arquivo temporário
MASCARA=$        # Caractere exótico para mascarar o separador na entrada de
                 # dados
#--------------------------[ Funções ]-------------------------------------

# o arquivo texto com o banco já deve estar definido
[ "$BANCO" ] || {
  echo "Base de dados não informada. Use a variável BANCO. "
  return 1
}

# verifica se o arquivo que o banco manipula tem permissão de leitura e
# escrita
[ -r "$BANCO" -a -w "$BANCO" ] || {
  echo "Base travada, confira as permissões de leitua e estrica"
  return 1
}

# Esconde ou mostra o caractere separador quando ele for literal
esconde() { tr $SEP $MASCARA ; }  # exemplo: troca : por §
mostra()  { tr $MASCARA $SEP ; }  # exemplo: troca § por :


# verifica se a chave primária existe
tem_chave() {
  grep -i -q "^$1$SEP" "$BANCO"
}

# verifica se a chave composta existe
tem_chave_2() {
  grep -i -q "^$1$SEP" "$BANCO"
}

# apaga o registro da chave $1 do banco
```

```
apaga_registro() {
   tem_chave "$1" || {
   echo "O registro '$1' não existe"
   return
   }
   grep -i -v "^$1$SEP" "$BANCO" > "$TEMP"    # apaga a chave
   mv "$TEMP" "$BANCO"                        # regrava o banco
   echo "O registro '$1' foi apagado"
}


# insere o registro $* no banco
insere_registro() {
  local chave=$(echo "$1" | cut -d $SEP -f1)  # pega primeiro campo
  echo "$*" >>"$BANCO"                        # grava o registro
  echo "Registro de '$chave' cadastrado com sucesso."
  return 0
}

# Exibe os nomes dos campos do banco, um por linha
campos() {
  head -n 1 "$BANCO" | tr $SEP \\n
}

# Exibe os dados do registro da chave $1
exibe_registro() {
  local dados=$(grep -i "^$1$SEP" "$BANCO")
  local i=0
  [ "$dados" ] || return                  # não achei
  campos | while read campo; do           # faz a leitura de cada campo
      i=$((i+1))                          # indice do campo
      echo -n "$campo: "                  # nome do campo
      echo "$dados" | cut -d $SEP -f $i | mostra  # conteúdo do campo
  done
}

# Exibe o valor do campo numero $1 do registro de chave $2 (opcional)
localiza_campo() {
  local chave=${2:-.*}
  grep -i "^$chave$SEP" "$BANCO" | cut -d $SEP -f $1| mostra
}

# Inclui ou atualiza a configuração de monitoramento no crontab
existe_crontab() {
 # verifica se já existe uma configuração
   grep -i -q "$1$" "$CRONTAB"
}

# Insere registro no crontab
insere_registro_crontab() {
 echo "$*" >>"$CRONTAB"
 echo "Configuração gravada com Sucesso"
 return 0
}
```