CENTRO UNIVERSITÁRIO UNIVATES

CURSO DE ENGENHARIA DA COMPUTAÇÃO

SISTEMA DISTRIBUÍDO PARA MONITORAR O USO DOS RECURSOS DE HARDWARE E SOFTWARE EM ESTAÇÕES DE TRABALHO GNU/LINUX

Jamiel Spezia

Monografia apresentada como parte das exigências para obtenção do grau de Bacharel em Engenharia da Computação.

Orientador: Maglan Cristiano Diemer

Lajeado, dezembro de 2007

AGRADECIMENTOS

Ao meu orientador Maglan Cristiano Diemer pela sugestão do tema, amizade, disponibilidade e orientação segura, fatores importantes que contribuíram para a realização deste trabalho.

À minha esposa Suzi por todo amor, dedicação, paciência e incentivo, que serviram de motivação para enfrentar as dificuldades encontradas ao longo deste ano. E com certeza você foi a pessoa mais importante para a conclusão deste trabalho.

Aos meus pais Singlair e Neli pelos anos de dedicação que contribuíram em muito para a minha formação pessoal e profissional. Ao meus irmãos Jonas e Régis que indiretamente me incentivam. À minha *nona* Santina que sempre procurou me mostrar o caminho certo.

Ao Alvaro, Sueli e Alvano por me receberem de braços abertos na família Ferrari e por entenderem a minha ausência neste ano.

Aos meus colegas pela troca de idéias, sugestões e pela força recebida. Ao Alexandre e William por lerem o trabalho e pelas opiniões construtivas.

RESUMO

Esta monografia tem por objetivo implementar um sistema distribuído para monitorar o uso dos recursos de hardware e software em estações de trabalho GNU/Linux. O trabalho descreve inicialmente conceitos sobre o monitoramento de recursos de hardware e softwares no sistema operacional GNU/Linux. Após, analisa as soluções existentes e justifica o desenvolvimento deste sistema. Por fim, descreve a implementação do sistema e analisa os resultados obtidos.

Palavras-chave: GNU/Linux, Monitoramento, Processos, Hardware, Software

ABSTRACT

This work aims to implement a distributed system to track the resource usage of hardware and software on GNU/Linux workstations. The work begins describing concepts about tracking hardware and software resources in the GNU/Linux operating system. After, analyzes existing solutions and justifies the development of this system. Finally, it describes the implementation of the system and analyzes the results.

Keywords: GNU/Linux, Tracking, Processes, Hardware, Software

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ASN.1	Abstract Syntax Notation 1
CPU	Central Processing Unit
FSF	Free Software Foundation
GCC	GNU Compiler Collection
GNU	GNU's Not Unix
GPL	General Public License
IP	Internet Protocol
ITU	International Telecommunication Union
ISO	International Standards Organization
LAN	Local Area Network
MIB	Management Information Base
OSI	Open Systems Interconnection
PID	Process Identification
POSIX	Portable Operating System Interface
SDMR	Sistema Distribuído para Monitoração de Recursos
SGBD	Sistema Gerenciador de Banco de Dados
SNMP	Simple Network Management Protocol
TCP	Transmission Control Protocol

TI	Tecnologia da Informação
UDP	User Datagram Protocol
WAN	Wide Area Network

SUMÁRIO

1 INTRODUÇÃO

No mundo dos negócios a palavra chave se tornou a maximização dos resultados, isto é, diminuir o tempo dos processos e aumentar o rendimento dos recursos. Para isso, torna-se essencial que as empresas e instituições explorem e utilizem os recursos providos pela Tecnologia da Informação TI. Entretanto, é fundamental que a empresa tenha informações relevantes sobre a utilização dos seus recursos de hardware bem como os softwares utilizados e como os mesmos são utilizados. Através desta informação pode-se melhorar o desempenho dos processos de TI com um melhor aproveitamento e alocação dos recursos de hardware disponíveis.

Para uma empresa ou instituição que possua um grande parque de máquinas torna-se complicado e lento o controle individual dos recursos de TI disponíveis. Assim, faz-se necessário o uso de ferramentas que tornem este controle automático e que disponibilizem essas informações de forma rápida, centralizada e acessível.

Existem várias soluções que fornecem informações como a produtividade dos funcionários, inventário, utilização de softwares e interação com máquinas clientes. No entanto, estas soluções são proprietárias e ou desenvolvidas para o sistema operacional Microsoft Windows. As soluções encontradas em software livre para o sistema operacional GNU/Linux focam suas funcionalidades na análise da rede, inventário e controle de configurações sem preocupar-se com uma análise detalhada dos softwares que são utilizados pelos usuários.

Visto que o sistema operacional GNU/Linux possui uma crescente adoção pelo mercado corporativo e que as soluções atuais para este sistema operacional não contemplam todos os recursos desejados, percebe-se que é de grande valia o desenvolvimento de uma solução que disponibilize as informações da utilização dos recursos de hardware e software nas estações de computadores em uma rede corporativa, visando a auxiliar a tomada de decisão, bem como a realocação de recursos, atualização de maquinário e monitoramento de processos no sistema operacional GNU/Linux.

Assim, o objetivo deste trabalho é desenvolver um sistema distribuído para monitorar o uso dos recursos de hardware e software em um parque de máquinas com o sistema operacional GNU/Linux. Para tanto, o Sistema Distribuído para Monitoração de Recursos - SDMR deverá capturar as informações em máquinas locais, enviá-las para processamento utilizando a estrutura de rede e armazená-las de forma centralizada em um banco de dados. O sistema deverá coletar informações referentes à utilização de processador, memória, disco rígido e temperatura do processador. Não serão implementadas as funcionalidades de inventário, porém tal

recurso poderá ser desenvolvido em trabalhos futuros ou integrado com outros sistemas existentes.

O próximo capítulo fornece embasamento teórico para o desenvolvimento do Sistema Distribuído para Monitoração de Recursos SDMR, enquanto que o Capítulo 3 analisa o funcionamento e características das soluções existentes no mercado. No Capítulo 4 descreve-se a implementação do SDMR, seguida da análise dos impactos causados pelo sistema no ambiente, no quinto capítulo . Por fim, apresenta-se os resultados obtidos, as conclusões e os trabalhos futuros que poderão constituir novas funcionalidades para o SDMR.

2 MONITORAMENTO DE RECURSOS DE HARDWARE E SOFTWARE

O Sistema Distribuído para Monitoração de Recursos - SDMR extrai informações sobre o uso dos recursos de hardware e software de estações de trabalho com o sistema operacional GNU/Linux em uma rede de computadores. Este capítulo aborda os conceitos fundamentais para entender o sistema. Os detalhes de implementação serão discutidos no Capítulo 4. Como ilustrado na Figura 1, o SDMR é composto por quatro elementos agente, coletor, servidor de banco de dados e console que serão detalhados a seguir.

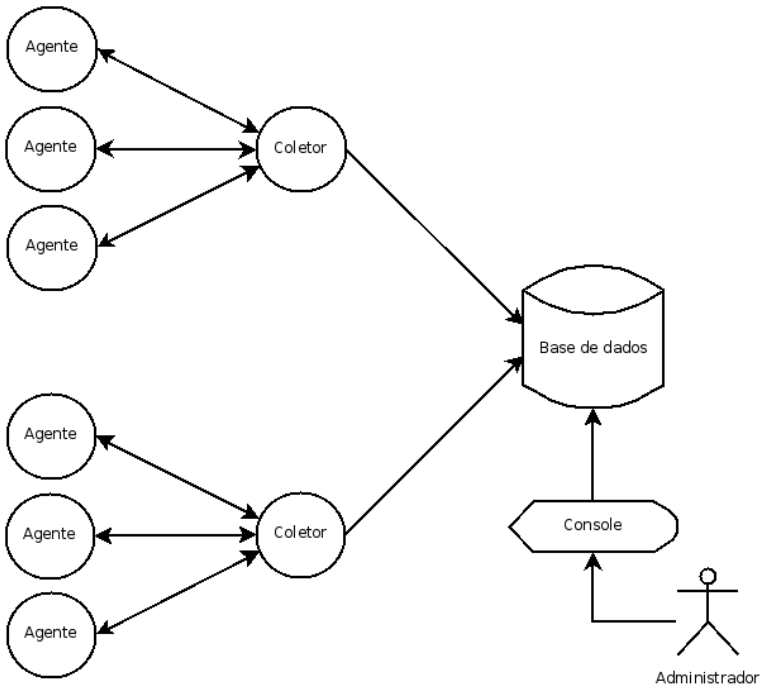


Figura 1 - Estrutura do sistema distribuído para monitoramento de recursos.

Cada estação de trabalho deve executar o processo agente. Este processo captura as informações de consumo de CPU, memória, temperatura do processador e partições do disco rígido. Enquanto o agente está executando, as informações por ele capturadas são temporariamente armazenadas na memória principal.

As informações extraídas por cada agente serão processadas por um coletor, que é responsável por requisitar/receber as informações de cada agente por ele controlado. Para isso, utiliza-se de um protocolo de aplicação específico para este fim. Mais detalhes sobre o protocolo serão discutidos na Seção 2.3.

As informações recebidas por cada coletor são armazenadas num banco de dados, o que permite que sejam realizadas estatísticas de uso de cada estação da rede de computadores. Armazenando as informações em um único servidor de

banco de dados, também facilita-se o acesso às informações coletadas, já que não há necessidade de se acessar cada agente no momento da análise dos dados. Por fim, o console é uma aplicação que disponibiliza de forma amigável as informações coletadas e armazenadas no banco de dados

Como descrito anteriormente, o coletor é o responsável por buscar, processar e armazenar as informações de vários agentes. Logo, deduz-se que um coletor possuirá um limite de agentes. Assim, adicionou-se na estrutura a possibilidade de ter vários coletores responsáveis por buscar, processar e armazenar as informações de seus respectivos agentes. Para a comunicação entre agente e coletor é utilizado o protocolo de aplicação SNMP (Simple Network Management Protocol), que tem a finalidade de monitorar e gerenciar uma rede de computadores.

Para que seja possível o entendimento sobre como o agente extrai informações do sistema operacional GNU/Linux, foram definidos, a seguir, conceitos sobre sistemas operacionais, GNU/Linux, processos, gerência de memória e métodos para obtenção de informação. Como o coletor requisita as informações do agente pela rede de computadores, torna-se necessário conceituar também protocolos de comunicação, em particular, o protocolo de aplicação SNMP e protocolo de transporte UDP. Por fim, conceituam-se bancos de dados, visto que as informações são armazenadas para consultas futuras.

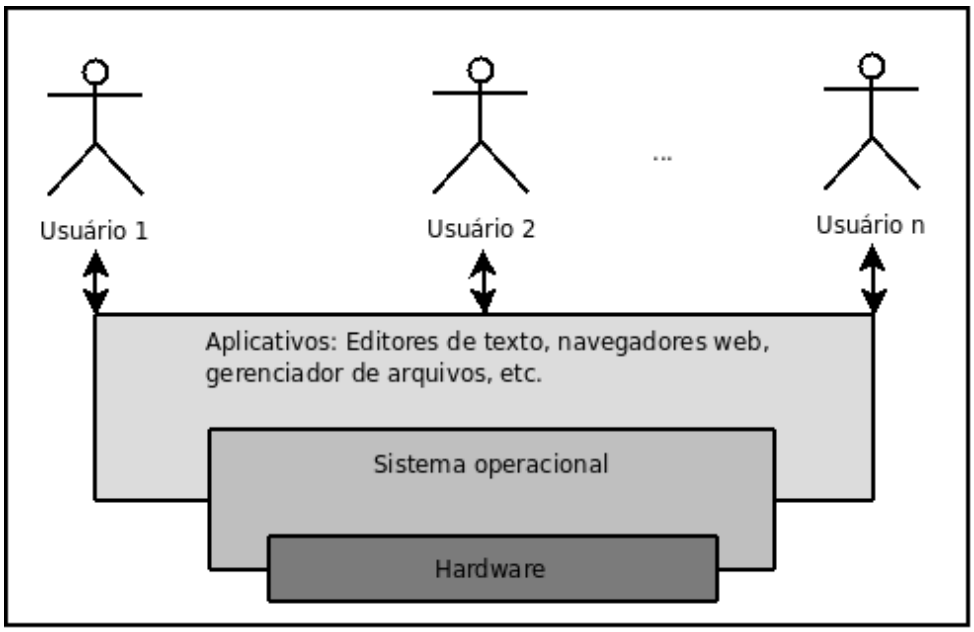
2.1 Sistema operacional

No conceito computacional, o computador é divido em hardware e software. Onde, o hardware é toda a parte física, ou seja, é formado por processador, memória, disco rígido, etc. Já os softwares são os aplicativos de usuário e o próprio sistema operacional.

O sistema operacional tem um papel importante para o bom funcionamento do hardware, sendo responsável por gerenciar a interação das aplicações do usuário e com hardware do computador (Oliveira, 2001).

Dentre as funcionalidades de um sistema operacional Oliveira (2001) cita a gerência de memória, escalonamento de processos, sistema de arquivos e controle dos dispositivos de entrada/saída.

Como ilustrado pela Figura 2, os aplicativos de usuário acessam os recursos do sistema operacional através de chamadas de sistema. As chamadas de sistema são funções implementadas no próprio sistema operacional. Sempre que necessário, os aplicativos executam estas funções passando parâmetros e aguardando o retorno do sistema operacional. A requisição de um arquivo, por exemplo, é feita através de uma chamada de sistema passando o nome do arquivo como parâmetro. O sistema operacional certifica-se da existência e retorna para aplicação o seu conteúdo (Oliveira, 2001).



Fonte: OLIVEIRA, Rômulo Silva de (2001, p. 2)

Figura 2 - Sistema computacional.

Segundo Oliveira (2001, p. 3), diversas informações sobre o estado do sistema são mantidas pelo sistema operacional . Estas informações são importantes para o próprio gerenciamento do sistema operacional. Entretanto, aplicativos de usuário podem usufruir destas informações para gerar relatórios e descobrir possíveis gargalos que impedem o aproveitamento máximo dos recursos de hardware.

Como exemplo de sistema operacional pode-se citar o Microsoft Windows, MacOS, AIX, HP-UX, FreeBSD, Linux, entre outros. Cada sistema operacional possui particularidades que influenciam diretamente na programação de aplicações. Como o SDMR é desenvolvido para obter as informações do sistema operacional GNU/Linux será estudado a seguir a forma como este sistema operacional trabalha com o gerenciamento de processos e memória.

2.2 Linux

Linux é um sistema operacional que surgiu, em 1991, idealizado por Linus Torvalds. Inicialmente foi desenvolvido para arquiteturas x86. Em sua versão beta, foi liberado para o desenvolvimento comunitário, que trouxe contribuições na implementação de outras necessidades, como por exemplo, para diversos dispositivos. Seu desenvolvimento é baseado nas especificações da POSIX[1] (Rodriguez, 2006).

Atualmente, o Linux suporta várias arquiteturas e nos últimos anos o seu uso na indústria, no meio acadêmico, no governo e também residencial aumentou significativamente. Possui o código fonte aberto e está licenciado sob a General Public License GPL[2], tornando-se um exemplo de sucesso em softwares de código fonte aberto (Rodriguez, 2006).

Rodriguez (2006) considera o Linux como sendo somente o núcleo ou *kernel*, e uma distribuição Linux como sendo o conjunto do *kernel*, ferramentas, interface gráfica e outros aplicativos. Moraes (2005) também utiliza estes termos e referencia GNU/Linux para uma distribuição instalada. Assim, dá-se os méritos ao projeto GNU is Not Unix GNU[3], que é mantido pela Free Software Foundation FSF e que, em conjunto com o Linux, fornam uma distribuição completa.

---

1 POSIX é um padrão que normaliza o desenvolvimento de um sistema operacional UNIX.

2 GPL é uma licença de uso utilizada por softwares livres. Site: http://www.gnu.org/licenses/gpl-3.0.html

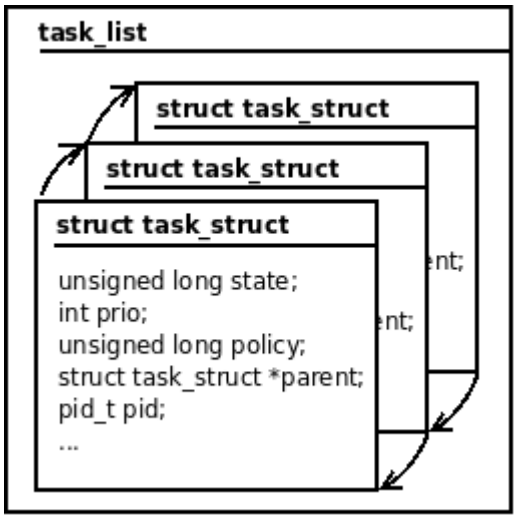
3 Projeto GNU foi iniciado em 1984 para desenvolver um sistema operacional completo, compatível com o Unix, que fosse software livre. Site: http://www.gnu.org

2.2.1 Processo

Para que seja possível monitorar os recursos de hardware e software é necessário obter informações referentes aos processos que rodam no sistema operacional. Segundo Rodriguez (2006) um processo é uma pequena instância de um programa. Um programa pode ser composto por um ou mais processos.

Para Oliveira (2001), um sistema operacional deve manter informações sobre os processos. Deste modo, o Linux mantém todos os processos em uma lista circular duplamente ligada, chamada de *task_list*. Cada elemento desta lista possui um descritor de processo.

O descritor de processos, também chamado de *task_struct*, é uma estrutura que mantém as informações de um único processo. Entre as várias informações estão o estado do processo, os sinais pendentes, os arquivos abertos, entre outros (Love, 2005).



Fonte: LOVE, Robert (2005, p. 25)

Figura 3 - *task_list* e o descritor de processos.

No Linux, o processo passa por um ciclo de vida de criação, execução e término (Oliveira, 2001). Para criar um processo utiliza-se a chamada de sistema *fork(),* que aloca recursos de hardware para o novo processo. Já a chamada de sistema *exit()* é utilizada para liberar os recursos alocados pelo processo (Love, 2005).

Cada processo é identificado por um número, também chamado de Process Identification PID. O PID é um número inteiro representado pelo tipo*pid_t* que, por padrão, pode chegar ao valor máximo de 32.768. No descritor de processos este número é armazenado no campo *pid* (Love, 2005).

Após a criação, o processo passa para execução e pode assumir vários estados. Os estados são descritos abaixo e ilustrados na Figura 4 (Oliveira, 2001).

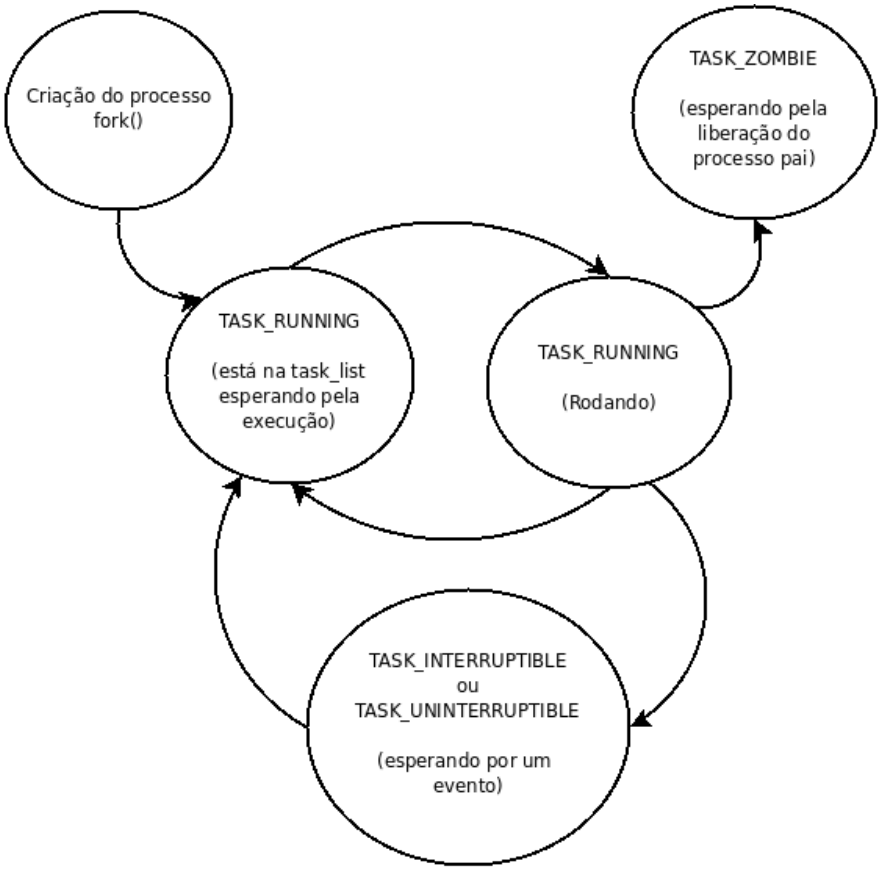
- **TASK_RUNNING**: está executando ou esperando para ser executado. O Linux possui um apontador para saber exatamente qual é o processo que está realmente executando. Os outros estão na lista esperando a execução.

- **TASK_INTERRUPTIBLE**: está bloqueado, esperando por uma condição, que pode ser, uma operação de entrada/saída, liberação de um recurso de sincronização ou uma interrupção de software. Ao ser estabelecida a condição, o processo volta para o estado TASK_RUNNING.

- **TASK_UNINTERRUPTABLE**: está bloqueado, esperando por uma condição crítica normalmente um evento de hardware e não pode sair deste estado até que o evento seja finalizado.

- **TASK_STOPPED**: pára a execução por ocorrência de certas interrupções de software. Ao receber outra interrupção, volta ao estado TASK_RUNNING. Este estado é geralmente utilizado por depuradores.

- **TASK_ZOMBIE**: estado que um processo filho assume logo após a sua execução completa. Fica neste estado até que o processo pai libere a alocação de seus recursos através da chamada de sistema *wait().*



Fonte: LOVE, Robert (2005, p. 28)

Figura 4 - Estados dos processos.

É possível saber o tempo de utilização do processador por cada processo através de variáveis mantidas na *task_struct*. Para isso o *kernel* possui um gerenciamento de tempo.

Para o gerenciamento do tempo o *kernel* trabalha com interrupções de tempo de hardware, que são definidas de acordo com a constante HZ. O HZ é definido no código fonte do *kernel* e seu valor padrão pode variar entre 100 a 1000 interrupções por segundo dependendo da arquitetura (Corbet, 2005).

Uma variável nomeada *jiffies* é criada durante a inicialização do sistema operacional. Seu valor é inicializado em 0 e incrementado em 1 a cada interrupção

de tempo. Assim, em um segundo ocorrem HZ interrupções de tempo que incrementam a variável *jiffies* neste mesmo valor. Com isso, pode-se deduzir que, ao dividir a variável *jiffies* pelo valor de HZ, obtém-se o tempo em segundos *(*Love, 2005). Exemplificando, se a constante HZ estiver definida em 100 interrupções de tempo por segundo, ao ocorrerem 500 interrupções de tempo, a variável *jiffies* armazenará o valor 500. Logo, dividindo-se o valor de *jiffies* por HZ obtém-se 5 segundos.

Cada processo possui variáveis que armazenam a quantidade de *jiffies* na *task_struct*. Com isso, o *kernel* incrementa estas variáveis a cada interrupção de tempo que o processo permaneceu utilizando o processador (Love, 2007).

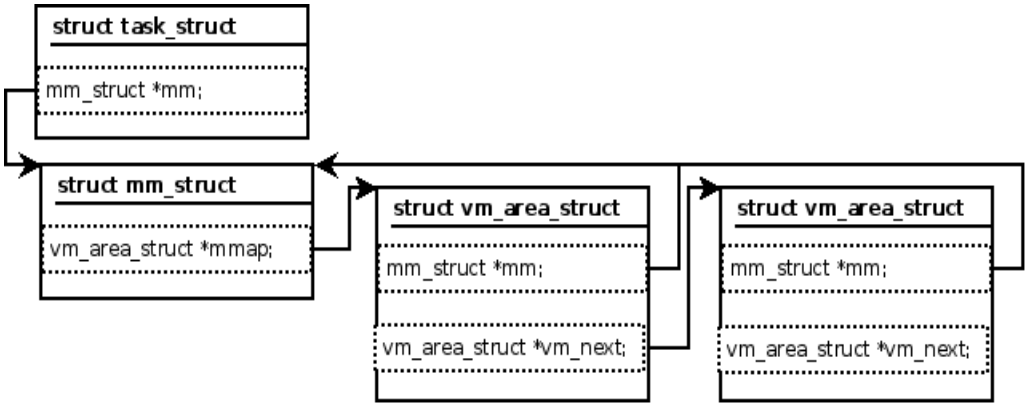
Coletando-se a informação de quantas interrupções um processo permaneceu utilizando o processador (*jiffies*) em um segundo, dividindo-a pelo número máximo de interrupções possíveis por segundo (HZ) e multiplicando-a por 100 obtém-se o percentual de uso do processador no momento especificado. Da mesma forma, para obter o valor em um período de 5 segundos é necessário coletar o número de *jiffies* executados neste período, dividí-los pelo número máximo de interrupções possíveis em 5 segundos (HZ*5) e multiplicá-lo por 100.

2.2.2 Gerenciador de memória

O gerenciador de memória é um subsistema do *kernel* que tem por objetivo alocar a memória física para um novo processo e liberá-la quando o processo deixar de existir (Rodrigues, 2006).

Os processos são alocados em páginas na memória física. Cada página possui um tamanho fixo que depende da arquitetura utilizada. Por exemplo, em arquiteturas de 32-bits utilizam-se páginas de tamanho 4 KB, enquanto que em arquiteturas de 64-bits utiliza-se 8 KB. Assim, uma máquina que possui tamanho de página de 4 KB e memória física de 1 GB tem 262.144 páginas. O Kernel possui um descritor de páginas que identifica quais podem ser realocadas (Love, 2005).

Quando um processo é criado o *kernel* aloca um intervalo de memória e o referência na *task_struct.* Este intervalo é definido pela estrutura *mm_struct* que faz uma referência para uma página em memória. Cada página é representada por uma estrutura *vm_area_struct*. Cada estrutura *vm_area_struct* faz uma referência à próxima página do intervalo alocado e uma referência de retorno à estrutura *mm_struct*. A Figura 5 ilustra a relação entre as estruturas (Rodriguez, 2006; Bovet, 2006).



Fonte: RODRIGUEZ, Claudia Salzberg. (2006, p. 227)

Figura 5 - Relação entre as estruturas.

Assim, o *kernel* mantém a informação de quantas páginas são alocadas por cada processo. Com isso é possível saber a ocupação de memória por um determinado processo. Multiplicando-se o valor de páginas pelo tamanho de cada página, obtém-se o valor em KB que um processo utiliza de memória.

2.2.3 Extração de informações do *kernel*

Conforme comentado anteriormente, Oliveira (2001, p. 3) cita que diversas informações sobre o estado do sistema são mantidas pelo sistema operacional . Há dois métodos para que o agente consiga capturar as informações mantidas pelo *kernel*. Os métodos são através das chamadas de sistemas e pelo acesso ao sistema de arquivos */proc* que serão detalhados a seguir.

2.2.3.1 Chamadas de sistemas

Chamadas de sistemas são funções que permitem a comunicação de aplicações de usuário com o *kernel* (Rodriguez, 2006). Com isso, é possível solicitar serviços ou informações ao sistema operacional.

Geralmente, ao utilizar uma linguagem de alto nível, as chamadas de sistema estão implementadas dentro de uma biblioteca. Assim, o programador utiliza a função oferecida pela linguagem e esta executa uma chamada de sistema para acessar a um determinado periférico (Oliveira, 2001).

Como mostrado na Figura 6, a aplicação do usuário executa a função *read()* da biblioteca de linguagem de programação que, por sua vez, executa a chamada de sistema ao espaço do *kernel*. O *kernel* faz o tratamento da chamada e retorna o valor para a biblioteca que a repassa para a aplicação do usuário.

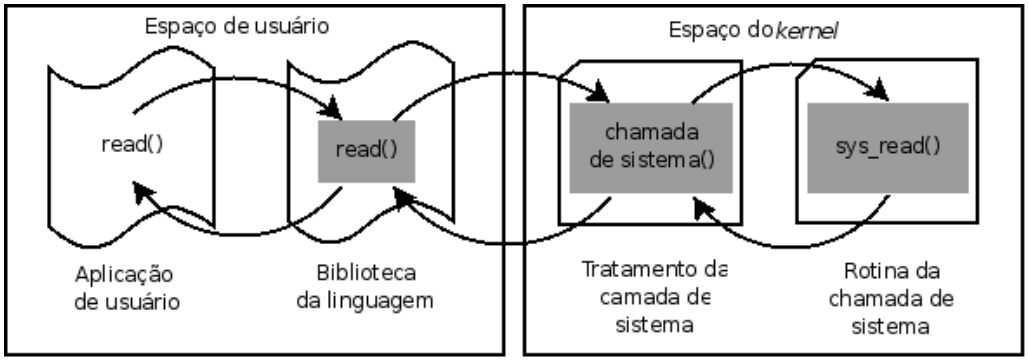


Figura 6 - Invocando uma chamada de sistema.

2.2.3.2 /PROC

O */proc* é um sistema de arquivos especial. Os arquivos encontrados no */proc* são acessados como qualquer outro arquivo do sistema, porém, não estão armazenados no disco rígido. O conteúdo dos arquivos não é estático e sim gerado pelo Kernel no momento da requisição de leitura (Mitchell, 2001).

As informações fornecidas possuem uma formatação de fácil interpretação humana. Por exemplo, ao visualizar o arquivo */proc/cpuinfo* obtém-se de forma clara as informações sobre a CPU (Mitchell, 2001). A Figura 7 mostra o conteúdo do arquivo *cpuinfo*, exibido através da aplicação *cat*[4].

```
% cat /proc/cpuinfo
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 5
model name      : Pentium II (Deschutes)
stepping        : 2
cpu MHz         : 400.913520
cache size      : 512 KB
fdiv_bug        : no
hlt_bug         : no
sep_bug         : no
f00f_bug        : no
coma_bug        : no
fpu             : yes
fpu_exception   : yes
cpuid level     : 2
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep
mtrr pge mca cmov pat pse36 mmx fxsr
bogomips        : 399.77
```

Fonte: MITCHELL, Mark. (2001, p. 148)

Figura 7 - Conteúdo fornecido pelo */proc/cpuinfo*.

Cada processo que roda no sistema GNU/Linux possui um diretório, nomeado com o PID, em */proc*. Estes diretórios são criados e removidos conforme os

4 O aplicativo *cat* exibe o conteúdo de um arquivo em um terminal.

processos são iniciados e finalizados (Mitchell, 2001). Assim, percorrendo a raiz do */proc* e capturando os diretórios com nome numérico obtém-se uma listagem de todos os processos que estão ativos no sistema. A Figura 8 ilustra os arquivos e diretórios que fazem parte do sistema de arquivos */proc*.

```
jamiel@eagle:~$ ls /proc
1      3     4848  5178  5550  6139  7091       devices      modules
10     35    4900  5179  5583  6141  7581       diskstats    mounts
10224  3554  4902  5249  5584  6144  7586       dma          mtrr
10243  3574  4923  5250  5644  6145  7596       dri          net
11     36    4939  5259  5683  6148  7612       driver       partitions
175    3631  4940  5276  5686  6170  7635       execdomains  scsi
2      37    4948  5356  5687  6173  7686       fb           self
204    3700  4949  5376  5905  6175  7699       filesystems  slabinfo
205    3702  4956  5395  5906  6183  7935       fs           stat
206    3703  4963  5407  5909  6184  8          ide          swaps
207    3704  4968  5408  5912  6188  845        interrupts   sys
2074   38    4971  5409  5914  6212  9          iomem        sysrq-trigger
2075   3806  4985  5410  5920  6216  9133       ioports      sysvipc
208    3807  4998  5411  5922  6422  9142       irq          tty
2113   4     5     5412  6     6424  9152       kallsyms     uptime
2114   4439  5013  5446  6103  6425  acpi       kcore        version
2115   4440  5031  5460  6105  6430  asound     key-users    version_signature
2265   4442  5032  5461  6107  6463  buddyinfo  kmsg         vmcore
2266   4445  5045  5462  6113  6877  bus        loadavg      vmstat
2451   4446  5117  5463  6125  6891  cmdline    locks        zoneinfo
2452   4447  5120  5464  6126  7     cpuinfo    meminfo
2651   4739  5161  5465  6137  7090  crypto     misc
```

Figura 8 - Arquivos e diretórios que fazem parte do sistema de arquivos */proc*.

Conforme Mitchell (2001), cada diretório de processo contém os seguintes arquivos:

- *cmdline*: contém a linha de comando completa do processo.

- *cwd*: é um link simbólico para o diretório de trabalho do processo.

- *environ*: contém as variáveis de ambiente do processo. As variáveis são separadas pelo byte nulo (\0).

- *maps*: contém informações sobre a região de memória e permissões de acesso.

- *root*: um link simbólico para o diretório raiz utilizado pelo processo.

- *stat*: fornece informações e estatísticas sobre o processo. A aplicação *ps* utiliza este arquivo para obter algumas informações.

- *statm*: fornece informações sobre o estado da memória em páginas.

- *status*: prove algumas informações referente aos arquivos *stat* e *statm* em uma formatação mais compreensível para os usuários.

O acesso ao */proc* facilita a obtenção das informações quando comparado com as chamadas de sistema (Linuxinsight, 2007). Isso, deve-se ao fato de que para obter a informação basta ler o arquivo e o /proc executará as devidas chamadas de sistemas para retornar a informação.

O agente pode ser desenvolvido de duas formas: como sendo um módulo ou uma aplicação de usuário. Um módulo nada mais é que um trecho de código, contendo funcionalidades, que pode ser incorporado como uma parte do *kernel*. Os módulos são carregados do espaço de usuário para o espaço do *kernel* (Moraes, 2005) e têm acesso a todas as funcionalidades do *kernel*, inclusive ao descritor de processos.

A aplicação de usuário é um programa que roda em modo usuário e não tem acesso direto às informações mantidas pelo *kernel*. Como visto anteriormente, uma aplicação de usuário utiliza as chamadas de sistema e/ou o sistema de arquivos */proc* para obter tais informações.

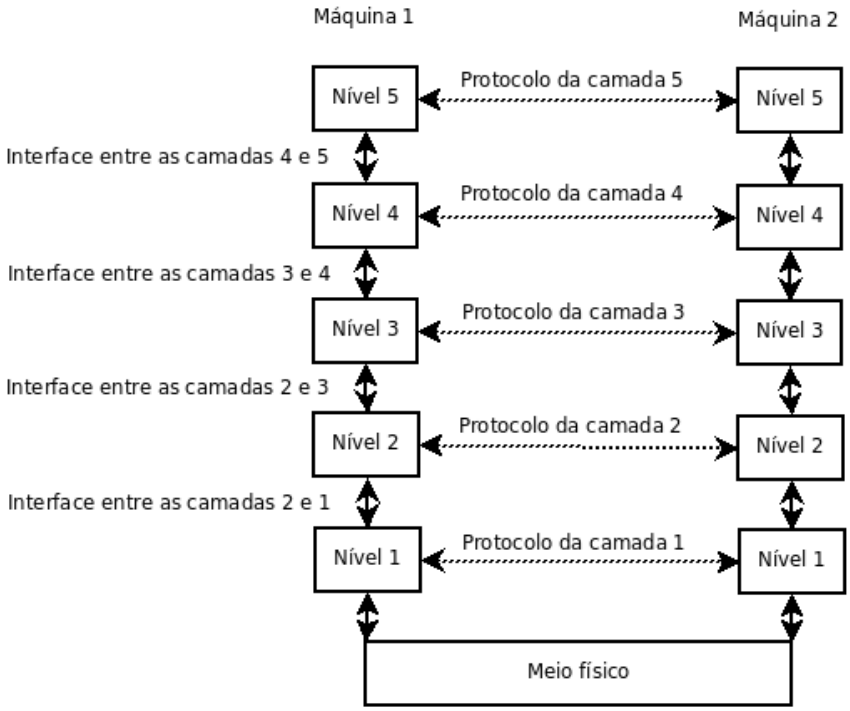
Ao desenvolver um módulo, deve-se ter um certo cuidado, pois o mesmo terá acesso a qualquer funcionalidade do *kernel* e um erro de programação ou a má utilização dos recursos pode impactar no sistema operacional de uma forma geral. Logo, o gerente de TI poderá ter uma certa desconfiança na hora de adotar uma aplicação que rode junto ao *kernel*.

Optou-se por desenvolver o agente como uma aplicação de usuário. Para obter as informações utiliza-se o sistema de arquivos /proc e em alguns casos chamadas de sistema. A forma de captura e a interpretação das informações são detalhadas no Capítulo 4.

2.3 Protocolo de comunicação

Para que haja troca de informações entre agente e coletor é necessário que os dois falem uma mesma linguagem e sejam capazes de manter uma conversação. Para isto, utilizam-se protocolos de comunicação que seguem um padrão e permitem a conectividade entre as máquinas.

Tanenbaum (1997, 19 p.) define protocolo como um conjunto de regras sobre o modo como se dará a comunicação entre as partes envolvidas . Como mostrado na Figura 9, os protocolos de comunicação são organizados em níveis e colocados um acima do outro. Cada nível de protocolo em uma máquina se comunica com o mesmo nível de outra máquina.



Fonte: TANENBAUM, Andrew S. (1997, p. 20)

Figura 9 - Organização conceitual dos protocolos em níveis .

Existem alguns modelos de referência que ditam regras de padronização para os níveis de protocolos. Por exemplo, o modelo TCP/IP e o modelo Open Systems Interconnection OSI (Tanenbaum, 1997).

Devido ao crescimento da Internet surgiu, em 1974, o modelo TCP/IP que visava a estruturação e resolução dos problemas com os protocolos até então

existentes. Este modelo possui quatro camadas definidas como host/rede (nível mais baixo), inter-rede, transporte e aplicação (nível mais alto) (Tanenbaum, 1997).

O modelo OSI surgiu baseado em uma proposta desenvolvida pela International Standardts Organization ISO. Este modelo possui sete camadas, definidas como camada física (nível mais baixo), enlace de dados, rede, transporte, sessão, apresentação e aplicação (nível mais alto) (Tanenbaum, 1997).

Os dois modelos tornam-se parecidos pois se baseiam no conceito de uma pilha de protocolos independentes. Apesar dessa semelhança os modelos têm muitas diferenças. Não é foco deste trabalho identificar as diferenças dos modelos, mas como Tanenbaum (1997, 43 p.) sugere, é possível consultar Piscitello (1993) para mais informações.

As próximas seções falam sobre as camadas de aplicação e transporte. Para isso, descreve-se um breve conceito e mostram-se opções de protocolos que podem ser utilizados para a comunicação entre o agente e o coletor.

2.3.1 Camada de aplicação

Este é o nível mais alto da pilha de protocolos, onde as aplicações que necessitam de uma interligação criam seu estilo de comunicação e passam-no para a camada abaixo que tratará de forma adequada a transmissão dos dados. Como

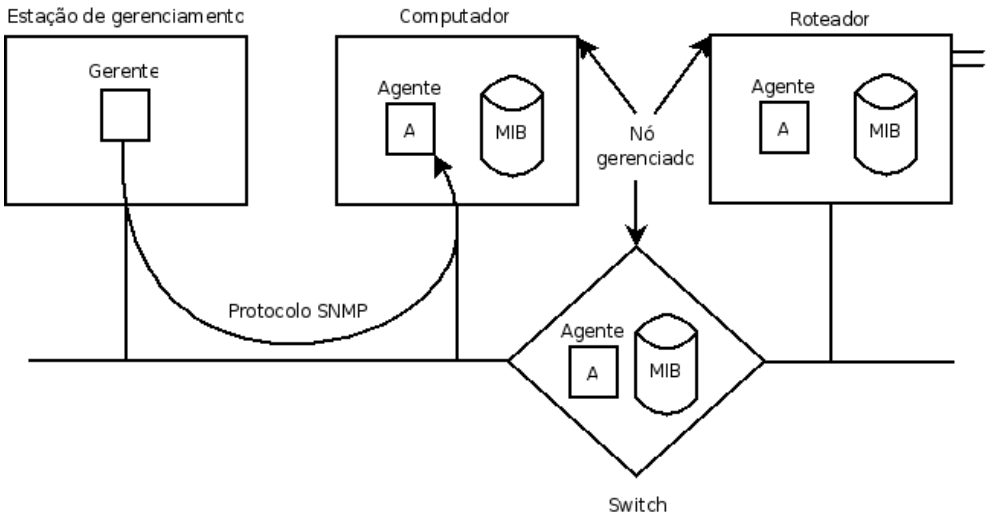
exemplo de protocolos de aplicação pode-se citar TELNET, FTP, SMTP, DNS, SNMP, HTTP, entre outros (Comer, 1998).

Uma aplicação de usuário pode criar seu próprio estilo de comunicação utilizando *sockets* e definindo um modelo de pacote próprio. Porém, para o SDMR pode ser utilizado o protocolo SNMP (Simple Network Management Protocol), pois o mesmo foi criado para monitorar e gerenciar uma rede de computadores (Tanenbaum, 1997).

Como mostrado na Figura 10, o modelo SNMP é composto por nós gerenciados, estações de gerenciamento e do protocolo SNMP. Os nós gerenciados podem ser computadores, roteadores, impressoras ou qualquer outro dispositivo capaz de comunicar informações para o mundo externo. Um nó gerenciado é composto por um agente SNMP que armazena informações do dispositivo local em uma estrutura de dados chamada de Management Information Base MIB. Por padrão, a MIB possui alguns objetos que armazenam valores sobre o sistema operacional, interfaces de rede e seu tráfego, estatísticas de pacotes IP, entre outros (Tanenbaum, 1997).

A estação de gerenciamento é um computador genérico portando um software especial que emite requisições para o agente SNMP e espera uma resposta. A estação de gerenciamento pode ser inteligente e executar processamentos sobre as informações obtidas. Dessa forma, o agente SNMP pode ser mais simples e ocupar o mínimo de recursos da máquina onde está hospedado (Tanenbaum, 1997).

A estação de gerenciamento interage com os agentes SNMP através do protocolo SNMP. Assim, é possível que a mesma consulte e altere o estado dos objetos de seus respectivos agentes (Tanenbaum, 1997).



Fonte: TANENBAUM, Andrew S. (1997, p. 720)

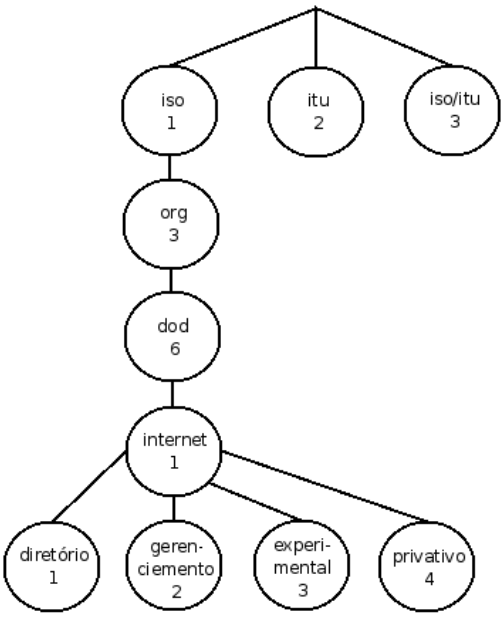
Figura 10 - Estrutura do modelo de gerenciamento SNMP.

Verifica-se que a estrutura do modelo SNMP torna-se parecida com a estrutura do SDMR. Assim, o agente do SDMR é um nó gerenciado que armazena na MIB informações dos recursos de hardware e software. O coletor torna-se uma estação de gerenciamento, que através do protocolo SNMP interage com os agentes.

Porém, a MIB não está estruturada para receber as informações propostas. Assim, necessita-se criar novos objetos que possam armazenar tais informações. Para isso, a próxima seção explica conceitos sobre a estruturação de uma MIB.

2.3.1.1 Estrutura e representação de objetos da MIB

Os objetos da MIB são organizados hierarquicamente em uma árvore administrada pela ISO e pela ITU. Com isso, o identificador de um objeto é a seqüência de rótulos numéricos ou textuais da raiz até o objeto em questão. A Figura 11 ilustra uma parte da hierarquia do identificador do objeto. Então, para acessar o objeto *gerenciamento* é possível requisitá-lo pelo identificador textual *iso.org.dod.internet.gerenciamento* ou pelo identificador numérico 1.3.6.1.2 (Comer, 1998).



Fonte: COMER, Douglas E.. (1998, p. 505)

Figura 11 - Parte da árvore de objetos.

O modelo de representação de um objeto é definido pela Abstract Syntax Notation 1 ASN.1. Assim, para criar um novo objeto é necessário defini-lo com a macro *OBJECT-TYPE* e informar quatro parâmetros. O primeiro parâmetro é *SYNTAX*, que define o tipo de dado que será armazenado no objeto. Os tipos de dados básicos são *INTEGER*, *BIT STRING*, *OCTET STRING*, *NULL* e *OBJECT*

IDENTIFIER. O segundo parâmetro é *MAX-ACCESS* e define o tipo de acesso permitido para a estação de gerenciamento. Os acessos mais comuns são leitura/escrita e somente leitura. O terceiro parâmetro é *STATUS* e identifica se a variável é atual, obsoleta ou desaprovada. O *DESCRIPTION* é o último parâmetro a ser informado e descreve para o usuário o que aquele objeto faz (Tanenbaum, 1997).

Um exemplo de declaração de objeto é ilustrado na Figura 12. O objeto é chamado de *pUtilizacaoDaCPU* e armazena o percentual de utilização da CPU. Este objeto é declarado na árvore de hierarquia com identificação 6 e localiza-se abaixo do objeto *tccEntradaParaProcessos*.

```
pUtilizacaoDaCPU OBJECT-TYPE
    SYNTAX      OCTET STRING
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "Armazena o percentual de utilização da CPU pelo processo
         que está radando na máquina local."
    ::= { tccEntradaParaProcessos 6 }
```

Figura 12 - Declaração do objeto *pUtilizacaoDaMemoria*.

2.3.1.2 O Protocolo SNMP

O protocolo SNMP define a comunicação entre o coletor e o agente. Para isso, são utilizadas sete mensagens. Seis das mensagens estão listadas na Figura 13 e a sétima mensagem é a mensagem de resposta (Tanenbaum, 1997).

Para requisitar um objeto, o coletor envia uma mensagem *get-request* passando o identificador do objeto a ser coletado. Ao receber a mensagem, o agente SNMP retorna a informação contida no objeto.

Mensagem	Descrição
Get-request	Solicita o valor de uma ou mais variáveis do nó gerenciado
Get-next-request	Solicita ao nó gerenciado a variável seguinte a atual
Get-bulk-request	Extrai uma tabela longa do nó gerenciado
Set-request	Atualiza uma ou mais variáveis do nó gerenciado.
Inform-request	Mensagem enviada entre estações de gerenciamento para descrever uma MIB local.
SnmpV2-trap	Relatório sobre traps que é enviado de um nó gerenciado para uma estação de gerenciamento.

Fonte: TANENBAUM, Andrew S. (1997, p. 734)

Figura 13 - Tipos de mensagens SNMP

É utilizada a versão SNMPv2, já que a mesma possui mais recursos e uma maior segurança quando comparada com a versão SNMPv1. Já a versão SNMPv3 foi descartada por consumir muitos recursos da rede.

Enfim, mencionado anteriormente, o protocolo de aplicação é apoiado pelo protocolo de transporte para transmitir os dados entre as estações. Este protocolo é implementado na camada de transporte, discutido a seguir.

2.3.2 Camada de transporte

A camada de transporte é responsável por prover a comunicação de um programa aplicativo de um ponto ao outro. Esta comunicação pode ocorrer de

maneira confiável, utilizando o protocolo TCP, ou de maneira imediata sem preocupação com a entrega, utilizando o protocolo UDP (Comer, 1998).

O protocolo UDP fornece conexões a vários programas aplicativos em um mesmo computador. Para isso, dispõe de um mecanismo de portas que diferencia os diversos programas executados em uma mesma máquina. O UDP utiliza o protocolo de rede IP para identificar o destino de um pacote. Porém, esta transmissão não é orientada à conexão, ou seja, o protocolo UDP não garante a entrega e nem a ordenação correta dos pacotes. Logo, os pacotes podem ser perdidos, duplicados ou entregues com problemas. Mas, por outro lado, este protocolo oferece a vantagem de entrega rápida e menor utilização de banda da rede, já que não há confirmação do recebimento dos pacotes (Comer, 1998).

Em redes locais (LANs), o protocolo UDP apresenta um bom funcionamento, já que as mesmas apresentam um pequeno atraso e são altamente confiáveis. Porém, esta vantagem não se torna válida ao se utilizar o protocolo em uma interligação de redes maiores (WANs) (Comer, 1998).

O protocolo TCP também utiliza portas para identificar o destino final em uma mesma máquina e faz uso do IP para identificar o destino dos pacote. O TCP é um protocolo orientado à conexão, ou seja, garante a entrega dos pacotes ao destinatário, e para isso, utiliza mensagens de confirmação de recebimento (ACK) e o conceito de janelas deslizantes. Logo, o protocolo TCP torna-se mais confiável que o protocolo UDP, mas possui a desvantagem de utilizar mais recursos da rede (Comer, 1998).

Por padrão, o protocolo de aplicação SNMP utiliza o protocolo de transporte UDP e a porta 161. Existe, porém, a possibilidade de alterar esta configuração (NetSnmp, 2007). Entretanto, optou-se por não alterar o valor padrão, já que o SDMR é utilizado em redes locais (LANs).

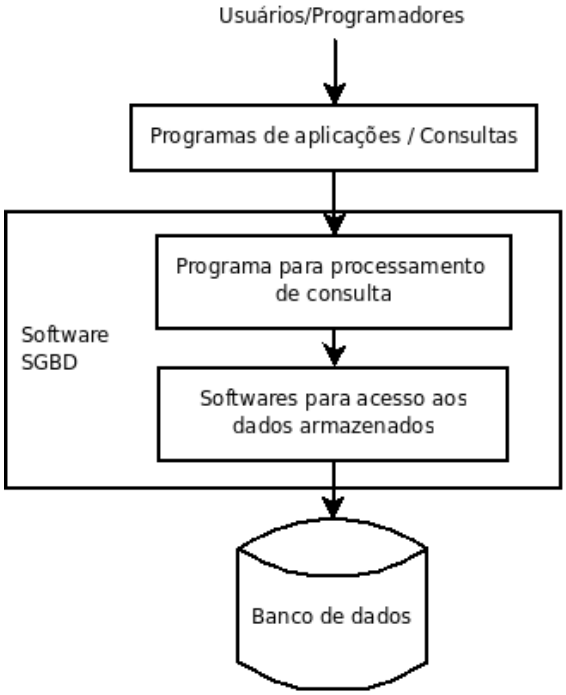
2.4 Banco de dados

De alguma forma, as informações extraídas das estações de trabalho devem ser armazenadas em um banco de dados centralizado. Para isso, define-se o conceito de banco de dados e de Sistema Gerenciador de Banco de Dados - SGBD.

Elmasri (2005, 4 p.) define banco de dados como uma coleção de dados relacionados e pode ser manipulado por um aplicativo ou por um SGBD.

O SGBD é um programa de propósito geral que facilita a construção, manipulação e compartilhamento de um ou mais bancos de dados entre os usuários e aplicações. O SGBD também é responsável pela proteção e segurança do banco de dados (Elmasri, 2005).

Como ilustrado na Figura 14, os usuários e programadores obtém acesso de leitura ou gravação de dados através de aplicações específicas que fazem os acessos ao banco de dados através do SGBD.



Fonte: ELMASRI, Ramez. (2005, p. 5)

Figura 14 - Configuração de um sistema de banco de dados simplificado.

O SDMR possui um único sistema gerenciador de banco de dados. Isto, deve-se ao fato de que os dados devem ser centralizados em um único ponto de acesso. Para o desenvolvimento foi adotado o SGBD *postgresql* (Postgresql, 2007). Os critérios utilizados para esta escolha foram a licença GPL e o conhecimento prévio da ferramenta.

Existem várias ferramentas para geração de relatórios, que permitem ao administrador acessar o SGBD e extrair as informações necessárias. Um exemplo de ferramenta é o Agata Report que é multiplataforma e tem suporte ao *postgresql* (Agata, 2007). Normalmente, para a utilização destas ferramentas o administrador tem que conhecer a modelagem do banco de dados. A modelagem do SDMR é descrita no Capítulo 4.

3 TRABALHOS RELACIONADOS

Neste capítulo é realizado um estudo individual das soluções existentes no mercado. Com isso, pretende-se mostrar as características, funcionalidades e arquitetura das soluções. Ao final, dá-se uma visão geral procurando expor uma análise comparativa entre as ferrametas.

3.1 NetEye

O NetEye surgiu em 2000 e estabeleceu, em 2005, uma parceria com a SADIG. O NetEye é uma solução que realiza auditorias nos computadores permitindo gerar estatísticas através de gráficos e relatórios (Neteye, 2007).

É possível monitorar a utilização de cada software por usuário, permitindo, assim identificar a forma como cada um desenvolve suas atividades. No relatório, visualiza-se detalhes das atividades realizadas, páginas acessadas, emails enviados e recebidos, arquivos utilizados, entre outros.

O software controla acessos indevidos informando ao administrador, através de alerta sonoro e visual, quando algum usuário acessa um página de Internet ou um programa não autorizado.

Além de monitorar a atualização de hardware, o NetEye, mantém um histórico dos softwares instalados disponibilizando a funcionalidade de atualização automática de software do parque de máquinas.

O programa ainda possibilita que o administrador assuma remotamente o controle das estações de trabalho. Com isso, é possível enviar mensagens para o usuário, executar comandos, reiniciar ou desligar as estações, exibir e fechar programas, copiar arquivos, capturar telas, suspender o login e bloquear o mouse e o teclado.

Não foram encontradas informações sobre a licença e plataformas suportadas. Também não foram encontrados locais para baixar o código fonte e ou a solução do NetEye. Assim, deduz-se que o sistema não é um software livre e não tem sua distribuição gratuita.

3.2 TraumaZero

O TraumaZero é desenvolvido pela empresa iVirtua Solutions. A empresa foi fundada em 2001 e provê serviços em soluções voltadas para o gerenciamento de

TI. O TraumaZero é uma solução que gerencia as áreas de infra-estrutura de TI, segurança da rede, serviços e informações (Ivirtua, 2007).

O programa utiliza uma sistemática para *backups* das informações e replicação de sistemas, podendo recuperar toda a estrutura de arquivos de uma unidade de disco através de cópia da imagem. Utiliza a tecnologia *multicast* para o envio simultâneo das imagens, podendo abranger ao mesmo tempo vários destinatários na rede.

Permite ao administrador acessar, monitorar e ter o controle dos computadores que fazem parte da rede. O acesso é feito com o auxilio de um navegador com interface *web*. Para maior segurança trabalha com autenticação assimétrica e utiliza criptografia de dados. Também gerencia o inventário de hardware e software provendo informações das modificações feitas em cada máquina.

A produtividade dos funcionários é exibida com o auxilio de gráficos e relatórios, informando ao administrador os acessos de cada software por usuário. Controla também os acesso indevidos a programas que possam causar danos ao sistema.

Ainda executa instalação, atualização e desinstalação de qualquer software em toda a rede ou em determinados computadores sem que o colaborador pare a atividade que está executando.

Ainda analisa a utilização de memória e processamento das tarefas diárias de cada computador. Com isso, é possível tomar decisões para realocação dos recursos e avaliar os investimentos.

O TraumaZero por ser utilizado nas plataformas Windows, DOS, OS2 e Linux. Analisando as características comerciais do site e a não disponibilidade do código fonte do produto, deduz-se que o TraumaZero é uma solução proprietária.

3.3 Cacic

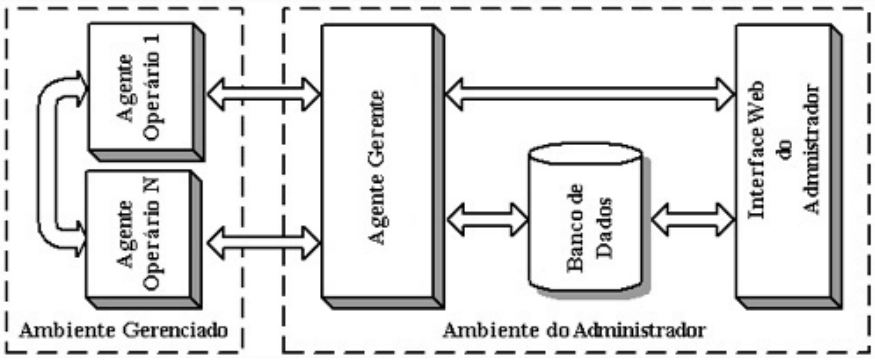
O Cacic é um software desenvolvido pela empresa DATAPREV e fornece um diagnóstico do parque computacional com informações como número de equipamentos, inventário de software e hardware, localização física dos equipamentos, entre outras (Cacic, 2007).

Tem por objetivo:

- Coletar informações sobre os componentes de hardware instalados em cada computador e disponibilizá-las aos administradores de sistemas;
- Alertar os administradores de sistemas quando forem identificadas alterações na configuração dos componentes de hardware de cada computador;
- Coletar diversas informações sobre os softwares instalados em cada computador e disponibilizá-las aos administradores de sistemas;

- Configurar programas em cada computador, de acordo com regras pré-estabelecidas pelos administradores de sistemas;
- Transferir arquivos para os computadores da rede, ocupando o máximo possível da largura de banda;
- Instalar novos softwares nos computadores gerenciados, tais como atualizações de programas ou patches de segurança;
- Identificar diretórios compartilhados considerados inseguros e aplicar as restrições de segurança necessárias;
- Coletar informações de Patrimônio (PIB, localização, etc.) de cada computador e disponibilizá-las aos administradores de sistemas;
- Alertar os administradores de sistemas quando forem identificadas alterações na localização física do computador;
- Permitir aos administradores de sistemas o envio de pequenas mensagens administrativas aos usuários de um computador específico ou usuários de um grupo de computadores.

Como ilustrado na Figura 15, o Cacic possui um ambiente administrador que comporta uma interface, um banco de dados e um agente gerente. O agente gerente tem por finalidade controlar as atividades realizadas no ambiente gerenciado, que por sua vez, é composto pelos agentes operários que coletam informações e comunicam-se com o agente gerente. Para minimizar custos e tempo de execução, os agente operários trocam as atividades entre si. Com isso, não é necessário requisitá-las ao agente gerente.



Fonte: Cacic (2007)

Figura 15 - Diagrama do funcionamento do CACIC

O Cacic está licenciado sob a licença GPL. Para utilizar o servidor do CACIC é necessário o sistema operacional Linux, base de dados MySQL, Apache e PHP. Os agentes do Cacic rodam nas versões 95, 95 OSR2, 98, 98 SE, ME, NT, 2000 e XP do Microsoft Windows e por enquanto não está disponível para Linux.

3.4 Puppet

A empresa Reductive Labs presta serviços de consultoria e assistência para o desenvolvimento do Puppet, que é uma linguagem declarativa para auxiliar administradores de sistemas na configuração dos computadores em uma rede. A linguagem é escrita pelo administrador de sistemas declarando quais tarefas devem ser executadas nas máquinas da rede (Puppet, 2007).

A linguagem permite executar diferentes fluxos de código dependendo do sistema operacional que está instalado no cliente. O programa possui vários recursos para auxiliar no processo de configuração. Porém ao se deparar com um

recurso não suportado pelo Puppet, o administrador poderá utilizar a função *exec*, que permite executar comandos externos.

O Puppet possui uma estrutura de servidor e clientes. Cada cliente contata periodicamente o servidor para verificar possíveis atualizações. Ao terminar a configuração, o cliente emite um relatório ao servidor comunicando-o sobre as alterações.

O Puppet pode ser utilizado nas plataformas Debian, RedHat, Solaris, SuSE, OS X, OpenBSD, CentOS e Gentoo. Está sob licença GPL.

3.5 Hyperic HQ

Hyperic HQ é uma solução desenvolvida pela empresa Hyperic. O Hyperic HQ foi projetado com o intuito de monitorar a infraestrutura de uma rede, controlando boa parte dos sistemas operacionais, servidores *web*, servidores de aplicação e servidores de base de dados. Disponibiliza interface *web* para monitorar, alertar, diagnosticar e controlar as aplicações (Hyperic, 2007).

O controle de inventário detecta aspectos do hardware e software, incluindo memória, processador, disco, dispositivos de rede, versões e informações sobre a configuração. O sistema detecta mudanças no inventário e alerta o administrador.

Também define políticas de segurança para auxiliar na detecção e registro dos acessos físicos e remotos em qualquer computador da rede.

O Hyperic HQ possui um servidor que recebe as informações, armazena e as disponibiliza para o administrador. Um agente instalado nos computadores tem por finalidade enviar para o servidor informações locais. O agente é projetado para ocupar o menor quantidade de memória e processador do computador onde está hospedado.

O Hyperic HQ suporta as plataformas Linux, Solaris (2.6 e superior), Windows (NT, 2000 e superior), HPUX 11.x, AIX (4.3 e superior), Mac OS X (10.4 e superior) e FreeBSD (5.x e 6.x). Está sob a licença GPL e tem seu código fonte aberto.

3.6 Zenoss

O Zenoss é desenvolvido pela comunidade e captura vários tipos de informações. Entre elas, pode ser citado o controle de eventos, desempenho, disponibilidade e informações de configuração (Zenoss, 2007).

A ferramenta percorre toda a rede e busca informações sobre memória, disco rígido, sistema operacional, serviços, processos e software. Assim, preenche a base de dados com o intuito de montar o inventário de hardware e software.

Também monitora a disponibilidade de serviços que rodam na rede. Os serviços podem ser cadastrados em uma interface e exibidos em listagens. Os serviços podem ser HTTP, SMTP, entre outros.

Além de verificar a performance dos dispositivos da rede, servidores e sistemas operacionais, o Zenoss monitora com o auxilio de gráficos e relatórios o uso total da CPU.

O programa possui a licença GPL e está disponível nas versões para Linux e Windows.

3.7 Análise comparativa

Ao analisar as soluções descritas acima conclui-se que há soluções que fornecem informações como a produtividade dos funcionários, inventário, utilização de softwares e interação com máquinas clientes. No entanto, estas soluções são proprietárias.

As soluções em software livre, sob licença GPL, focam suas funcionalidades na análise da rede, inventário e controle de configurações. No caso específico do Zenoss existe a possibilidade de se obter informações sobre o uso total da CPU, porém não há um detalhamento específico do uso de CPU e memória por processos.

Em uma análise superficial e genérica da arquitetura de funcionamento das soluções, nota-se que há um agente instalado em cada máquina. Estes coletam as informações locais e as concentram em um servidor para análise futura. No caso específico do Cacic, os agentes procuram trocar informações entre si com o intuito de minimizar os custos de acesso ao servidor.

No capítulo seguinte, explica-se a implementação do SDMR, que segue a idéia explanada no parágrafo anterior onde os sistemas possuem agentes locais para captura de informações e um servidor único para armazená-las.

4 IMPLEMENTAÇÃO

Este capítulo trata da implementação do Sistema Distribuído para Monitoração de Recursos SDMR. O sistema é desenvolvido utilizando a linguagem C (Tenenbaum, 1995; Mizrahi, 1990) e compilado com a ferramenta *gcc* (GCC, 2007). Todo o código fonte do SDMR encontra-se nos apêndices deste trabalho.

Para descrever o SDMR apresenta-se, a seguir, a forma de captura das informações na estação de trabalho, como acontece a troca de informações entre o agente e o coletor, o processamento das informações pelo coletor antes de armazená-las na base de dados, a estrutura da base de dados, comentários sobre o console e por fim o processo de compilação e execução do SDMR.

4.1 Agente

Com o intuito de ocupar o mínimo de recursos da máquina onde está hospedado, o agente tem por objetivo único extrair as informações sem se

preocupar com o processamento das mesmas. As informações são armazenadas na estrutura da MIB em memória principal, evitando assim o acesso ao disco e melhorando o desempenho do agente em relação ao processamento. Para alocar menos memória são gravados somente os processos que possuem um processamento maior que zero *jiffies* no intervalo de tempo avaliado.

Para extrair as informações de processamento, memória e temperatura utiliza-se o sistema de arquivos */proc*. Para a escolha deste método foram utilizados os critérios de conhecimento prévio, fácil entendimento e o fato de programas como *ps, top e htop* utilizarem este mesmo método (GNU, 2007; Htop, 2007). Já as informações de particionamento do disco são obtidas através de chamadas de sistema.

Como visto na Seção 2.2.3.2, cada processo possui um subdiretório no diretório /proc onde disponibilizam estas informações*.* Dentro de cada subdiretório, os arquivos são compostos de uma única linha com informações separadas por espaço. O formato do arquivo dificulta um pouco a leitura para humanos, porém é adequado para utilização em nível de programação (Mitchell, 2001). As informações de processamento e memória são extraídas dos arquivos *stat* e *statm.*

O arquivo *stat* contém 42 informações, sendo que as extraídas pelo agente são: (Mitchell, 2001)

- Primeira informação: identifica o id do processo. Na Figura 16 é identificado pelo valor 6072.

- Segunda informação: nome do arquivo executável. Na Figura 16 é identificado pelo valor (soffice.bin).

- Décima quarta informação: número de *jiffies* que o processo executou em modo usuário. Na Figura 16 é identificado pelo valor 1258.

- Décima quinta informação: número de *jiffies* que o processo executou em modo *kernel*. Na Figura 16 é identificado pelo valor 63.

```
jamiel@eagle:~$ cat /proc/6072/stat
6072 (soffice.bin) S 5831 5747 5747 0 -1 4202496 24580 82 533 0 1258 63 0 0 15 0
 6 0 17918 294838272 22340 4294967295 134512640 134856208 3219435648 3219434204
4294960144 0 0 4096 2076271871 4294967295 0 0 17 1 0 0 0
```

Figura 16 - Visualização do arquivo *stat*.

Como na Figura 16, em uma leitura do arquivo *stat* obtêm-se dois valores de *jiffies*. A informação 1258 é incrementada enquanto o processo executa o processador em modo usuário e o valor 63 enquanto executa em modo kernel. Nos cálculos essas informações são somadas e tornam-se um único valor de *jiffies*.

Para calcular o percentual de ocupação do processador por um processo deve-se fazer duas leituras do valor de *jiffies* em tempos diferentes e utilizar a equação $p=\frac{(jiffies'-jiffies)}{(HZ*(t'-t))}*100$, onde, p é o percentual de ocupação do processador, *jiffies'* é a quantidade de *jiffies* no tempo *t'*, *jiffies* é a quantidade de *jiffies* no tempo *t* e HZ é a constante que define o número máximo de *jiffies* em um segundo.

Para exemplificar, supõem-se que a primeira leitura no tempo zero segundos é de 1000 *jiffies,* a segunda leitura no tempo 32 segundos é de 3800 *jiffies* e o valor da constante HZ é definida em 100 interrupções por segundo. Assim, faz-se o seguinte cálculo, $p=\frac{(3800-1000)}{(100*(32-0))}*100=87,5$ e conclui-se que o percentual de ocupação do processador pelo processo em questão é de 87,5% no intervalo de 32 segundos.

Como ilustrado na Figura 17, o arquivo *statm* contém somente informações sobre o estado da memória. É composto por 7 informações, sendo que o agente utiliza somente a segunda. Essa informação identifica o tamanho em páginas da memória residente ocupada pelo processo (Mitchell, 2001). Como visto no Seção 2.2.2, o tamanho de cada página é definido em 4 KB. Para obter este valor utiliza-se a função *sysconf(_SC_PAGESIZE)* e divide-se o valor por 1024 (GNU, 2007).

```
jamiel@eagle:~$ cat /proc/6072/statm
75876 23437 16994 84 0 21746 0
```

Figura 17 - Visualização do arquivo *statm*.

Agora é possível saber a quantidade de memória ocupada pelo processo multiplicando-se o valor obtido no *statm* pelo tamanho da página em KB. Para obter o percentual de ocupação de memória do processo pode-se utilizar a equação $p=\frac{memOcupada}{totalDeMemória}*100$. Onde, p é o percentual de ocupação da memória pelo processo, *memOcupada* é o valor em KB que o processo utiliza de memória e *totalDeMemória* é o valor em KB de memória instalada na máquina. O valor *totalDeMemória* é obtido pelo arquivo *meminfo* que está localizado na raiz do diretório */proc*.

Para extrair as partições montadas no sistema utilizou-se a chamada de sistema *getmntent()*. Após ter os dados das partições foi possível obter, com o auxilio da chamada de sistema *statfs()*, o número de blocos totais da partição, número de blocos livres e tamanho do bloco. Então, para saber o número de blocos usados subtraiu-se os blocos totais dos blocos livres e para converter os resultados em KB multiplicou-se o número de blocos pelo tamanho do bloco e dividiu-se por 1024.

Já a coleta da temperatura é mais simples, o arquivo lido para computadores com suporte a *acpi* é o /proc/acpi/thermal_zone/TZ00/temperature. O caminho do arquivo é identificado nas configurações do agente. Com isso, é possível ajustar o caminho do arquivo, caso o suporte a esta funcionalidade seja disponibilizado em outro local. Se o computador não tiver suporte deve-se comentar a linha no arquivo de configuração.

O intervalo de captura para cada informação é definido no arquivo de configuração do agente. Por exemplo, pode-se capturar informações dos processos a cada 2 segundos, de temperatura a cada 60 segundos e de partições a cada 360 segundos. Detalhes sobre o arquivo de configuração são encontrados na Seção 4.5.

Como comentado anteriormente, os dados capturados são armazenados na estrutura da MIB. Esta estrutura é definida a seguir e o código fonte encontra-se no Apêndice ª

4.2 Protocolo

Nas estações de trabalho é instalado o agente do SNMP, nomeado *snmpd*. O agente do SNMP é responsável por extrair informações da estação de trabalho, armazená-las em uma MIB e responder às requisições das estações de gerenciamento.

Porém, o agente SNMP não possui uma MIB que gerencie as informações propostas no trabalho. Assim, além de coletar as informações, o agente SDMR estende o *snmpd* e registra uma nova MIB para agregar as informações propostas.

A MIB registrada pelo agente SDMR é chamada de *tcc.* Este novo objeto está localizado abaixo do objeto *netSnmp* e possui o identificador 1000. Abaixo do objeto *tcc* encontra-se o objeto *tccTabela*. A seguir, foram criados os objetos *tccProcessTable, tccTemperatureTable* e *tccPartitionTable,* que são uma seqüência de informações especificadas pelo objeto abaixo de sua estrutura. Este objeto, por sua vez, identifica um nodo de informações. Como ilustrado na Figura 18, o objeto *tccTemperaturaTable* é uma seqüencia do objeto *tccEntradaParaTemperatura,* que identifica o nodo e armazena as informações através dos objetos *tIDDaLinha*, *tDataHoraDaColeta* e *tTemperatura*.

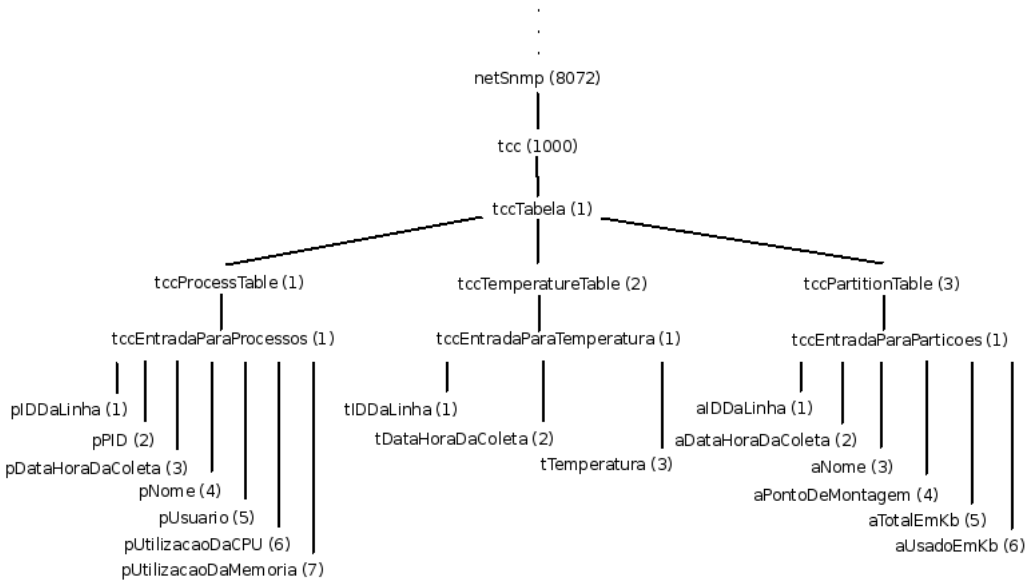


Figura 18 - Visualização em árvore da MIB *tcc*.

O agente SDMR grava as informações extraídas em nodos nas seqüências. Cada seqüência possui uma variável de controle que armazena o número do próximo nodo a ser utilizado.

No coletor é implementada uma estação de gerenciamento capaz de interagir com os agentes através do protocolo SNMP (Netsnmp, 2007). Assim, é possível capturar as informações enviando uma mensagem de requisição e passando o identificador do objeto a ser coletado. Para isso, o coletor requisita um objeto de cada vez. Assim, para requisitar as informações de um nodo da seqüência *tccProcessTable*, por exemplo, devem ser executadas sete requisições, ou seja, uma requisição para cada objeto do nodo.

O agente é capaz de identificar cada requisição. Assim, libera-se a memória ocupada por um nodo após a leitura completa do mesmo. Quando não há mais nodos a serem lidos, o agente reinicia a variável de controle dos nodos. Ou seja, a

cada final de coleta é liberada toda a memória ocupada pelas informações capturadas e recomeçado a preencher novamente a seqüência a partir do nodo zero.

Para implementação desta etapa é utilizada a API do NetSnmp. A API fornece funções para registrar uma MIB, manipular as informações e para comunicação entre o agente e coletor (Netsnmp, 2007).

4.3 Coletor

O coletor é responsável por requisitar as informações dos agentes e agrupá-las antes de gravar na base de dados. Como comentado anteriormente, o coletor utiliza o protocolo SNMP para enviar uma requisição e receber as informações dos agentes. O agrupamento das informações é importante para diminuir o número de registros gravados na base de dados. Isso deve-se ao fato de que é pouco interessante saber a taxa de processamento no intervalo de segundos, e sim no intervalo de minutos, horas, dias ou até mesmo meses.

O coletor faz a requisição das informações para as máquinas que estiverem cadastradas no arquivo de configuração. As requisições são feitas em seqüência, ou seja, o coletor requisita as informações de um agente por vez.

Pode ser que o coletor não consiga obter a informação de um agente. Uma das causas possíveis é o estouro do tempo de espera (*timeout*). Isso acontece

quando a estação de trabalho está desligada ou o agente SNMP não está rodando. Nesse caso, o coletor faz a requisição para o próximo agente.

Há um parâmetro que especifica o intervalo de tempo para iniciar novamente a coleta de todos os agentes. Este intervalo é subtraído com o tempo de requisição/processamento, ou seja, se o coletor estiver configurado para requisitar as informações a cada 30 minutos e levar um tempo de requisição/processamento igual a 10 minutos, ficará 20 minutos ocioso até a próxima requisição de coleta.

Quando o coletor recebe as informações de um agente dá-se início ao agrupamento das informações de processamento e memória. As informações são agrupados por PID, nome do processo e dono do processo.

Na Figura 19 tem-se informações de três capturas em uma máquina. Então, para agrupar as informações do percentual de CPU soma-se os valores de um processo e divide-se o resultado pelo número de tempos. Por exemplo, para agrupar o processo 5876 faz-se o cálculo $\frac{(6+12)}{3}=6$.

dataHoraDaColeta	PID Nome	Dono	% CPU	% Memória	
20071116 140500	5874 Xorg	root	10	30	Tempo 1
20071116 140500	5876 kmail	jamiel	6	7	
20071116 140502	5874 Xorg	root	20	30	Tempo 2
20071116 140502	5876 kmail	jamiel	12	7	
20071116 140504	5874 Xorg	root	6	30	Tempo 3
20071116 140504	5987 kopete	jamiel	6	3	

Figura 19 - Informações dos processos antes do agrupamento.

O agrupamento da ocupação de memória é diferente, já que, as informações coletadas são de processos que tiveram um processamento maior que zero. Logo,

soma-se o valor do percentual de memória e divide-se o resultado pelo número de ocorrências do processo. Por exemplo, para agrupar o processo 5876 faz-se o cálculo $\frac{(7+7)}{2}=7$.

Para cada linha agrupada são armazenadas duas informações de tempo que identificam o período do agrupamento. Este período é formado pela primeira e última data e hora de coleta encontrada na lista de processos não agrupados. Assim, pode-se saber que o processo 5876 ocupou 6% de processamento e 7% da memória no dia 16/11/2007 das 14:05:00 até as 14:05:04. A Figura 20 mostra como foram agrupadas as informações mostradas na Figura 19.

dataHoraInicial	dataHoraFinal	PID	Nome	Dono	% CPU	% Memória
20071116 140500	20071116 140504	5874	Xorg	root	12	30
20071116 140500	20071116 140504	5876	kmail	jamiel	6	7
20071116 140500	20071116 140504	5987	kopete	jamiel	2	3

Figura 20 - Informações dos processos agrupados.

Os dados de temperatura e partições não são agrupados, já que, estas informações não mudam freqüentemente como os dados dos processos. Para diminuir o volume destas informações é possível alterar o intervalo de tempo de captura do agente.

A identificação do nome do computador é feita através do coletor. No arquivo de configuração é necessário adicionar o IP do computador a ser monitorado e um nome para o mesmo. Então depois de coletar as informações o coletor as relaciona com o nome da máquina, que deve ser único.

Por fim, o coletor grava essas informações na base de dados. A estrutura da base é descrita a seguir.

4.4 Base de dados

As informações são armazenadas em um Sistema Gerenciador de Banco de Dados SGBD instalado em uma única máquina, centralizando as informações coletadas em um único ponto de acesso.

Para o desenvolvimento foi adotado o SGBD postgresql (Postgresql, 2007). Foram utilizados os critérios de licença e conhecimento prévio para a escolha. Caso seja necessária a utilização de um outro SGBD deve-se somente reescrever as funções de conexão com o banco e execução dos SQLs do coletor.

Como mostrado na Figura 21, a estrutura do banco de dados foi modelada em três tabelas: *processos*, *particoes* e *temperaturas*. Para evitar problemas com a codificação de caracteres não foram utilizados acentos para nomes de tabelas e campos.

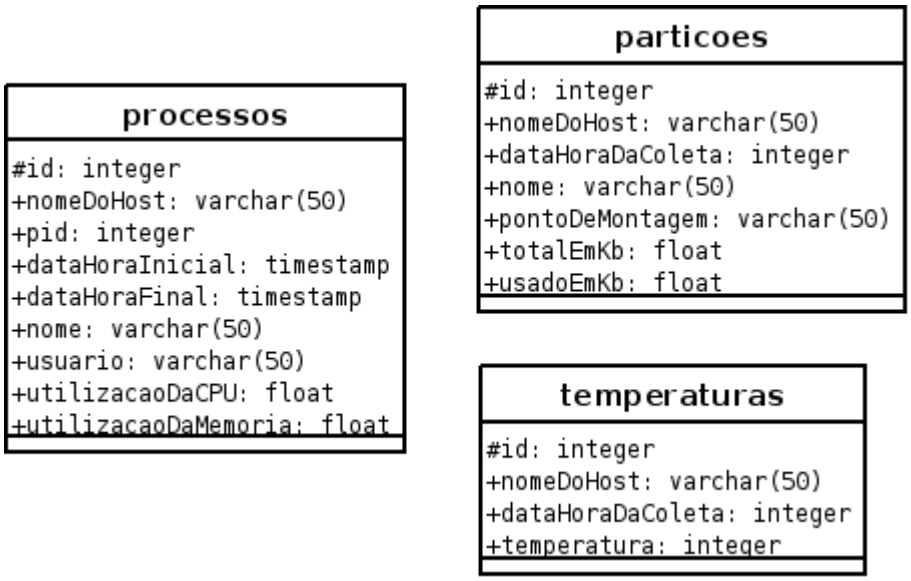


Figura 21 - Estrutura da base de dados.

A tabela *processos* armazena as informações referentes à utilização do processador e memória pelos processos. Nesta tabela é possível diferenciar qual é a máquina e/ou usuário que rodou determinado processo. Segue abaixo a descrição dos campos da tabela *processos*:

- *id*: campo de valor numérico e seqüencial. O valor é definido pelo banco no momento da inserção do registro.

- *nomeDoHost*: identifica o nome da máquina da qual foi coletada a informação sobre o processo. O nome do host é especificado no coletor.

- *pid*: campo de valor numérico que armazena o número de identificação do processo coletado PID.

- *dataHoraInicial* e *dataHoraFinal*: estes campos armazenam a data e hora, e em conjunto especificam o período de coleta da informação. Estas informações são importantes para a geração dos relatórios. Com isso é possível, por exemplo, saber a média das informações coletadas em 30 dias.

- *nome*: campo de valor textual que armazena o nome do processo.

- *usuario*: campo de valor textual que armazena o dono do processo, ou seja, o usuário que utilizou o processo.

- *utilizacaoDaCPU*: percentual de utilização da CPU no período especificado por *dataHoraInicial* até *dataHoraFinal.*

- utilizacaoDaMemoria: percentual de utilização da memória no período especificado por *dataHoraInicial* até *dataHoraFinal.*

A tabela *particoes* armazena toda a estrutura de partições de uma determinada máquina e, para cada partição, armazena as informações de espaço total e espaço utilizado. Segue abaixo a descrição dos campos da tabela *particoes*:

- *id*: campo de valor numérico e seqüencial. O valor é definido pelo banco no momento da inserção do registro.

- *nomeDoHost*: identifica o nome da máquina da qual foi coletada a informação sobre a partição. O nome do host é especificado no coletor.

- *dataHoraDaColeta*: armazena a data e hora do momento em que foi extraída a informação na máquina monitorada. Toda a tabela de particionamento de um disco será extraída no mesmo momento. Com isso, é possível identificar quais partições fazem parte de uma mesma máquina.

- *nome*: campo de valor textual que armazena o nome da partição.

- *pontoDeMontagem*: campo de valor textual que armazena o ponto de montagem.

- *totalEmKb*: campo de valor numérico que contém o espaço total em KB da partição.

- *usadoEmKb*: campo de valor numérico que contém o espaço utilizado em KB da partição.

Para obter o percentual de uso de uma determinada partição basta utilizar a equação $p=\left(\frac{usadoEmKb}{totalEmKb}\right)*100$.

A tabela *temperaturas* armazena a temperatura do processador em um determinado momento. Segue abaixo a descrição dos campos que compõem esta tabela:

- *id*: campo de valor numérico e seqüencial. O valor é definido pelo banco no momento da inserção do registro.

- *nomeDoHost*: identifica o nome da máquina da qual foi coletada a informação sobre a temperatura. O nome do host é especificado no coletor.

- *dataHoraDaColeta*: armazena a data e hora do momento em que foi extraída a informação na máquina monitorada.

- *temperatura*: campo de valor numérico que armazena a temperatura do processador no tempo especificado pelo campo *dataHoraDaColeta*.

O acesso às informações pode ser feito através da aplicação em linha de comando *psql* ou pela aplicação gráfica *pgAdmin3* (Postgresql, 2007). Também é possível desenvolver programas ou utilizar programas existentes que fazem o acesso a esta base de dados e retornam relatórios de uma forma mais amigável e simples.

4.5 Parametrizações

Tanto o agente quanto o coletor possuem arquivos de configuração que permitem principalmente a escolha do intervalos de tempo de captura. Para a implementação foi utilizada a biblioteca *libconfuse* que auxilia na leitura dos arquivos de configurações (Libconfuse, 2007).

A Figura 22 mostra o arquivo de configuração do agente. Nele é possível ajustar os tempos em segundo para o intervalo de captura de cada informação, informar o arquivo para a captura do valor de temperatura e permitir a exibição de notícias. Os parâmetros são:

```
# Arquivo de configuração do agented

# Configuração do agente
agtTempoDeExecucaoDoLaco = 2
agtTempoDeCapturaParaProcesso = 2
agtTempoDeCapturaParaTemperatura = 600
agtTempoDeCapturaParaParticao = 3600

agtProcTemperatura = /proc/acpi/thermal_zone/TZ00/temperature

# 1 exibe noticias
agtExibeNoticia = 0
```

Figura 22 - Arquivo de configuração do agente.

- *agtTempoDeExecucaoDoLaco*: especifica o intervalo de tempo em segundos que o agente deve checar as variáveis de controle para saber se há uma nova captura de informações. Este número deve ser menor ou igual ao menor número informado nos tempos de captura.

- *agtTempoDeCapturaParaProcessos*: especifica o intervalo de tempo em segundos para a captura das informações de processos.

- *agtTempoDeCapturaParaTemperatura*: especifica o intervalo de tempo em segundos para a captura da informação de temperatura do processador.

- *agtTempoDeCapturaParaParticao*: especifica o intervalo de tempo em segundos para captura a das informações referentes as partição do disco rígido.

- *agtProcTemperatura*: arquivo para obter a informação de temperatura do processador. Este caminho pode variar de máquina para máquina. Em máquinas antigas não é possível obter esta informação. Neste caso, deve-se comentar esta linha.

- agtExibeNoticias: utiliza-se este parâmetro para exibir notícias que facilitam o debug do código fonte. O valor 1 habilita a exibição das mensagens e é utilizado durante o desenvolvimento.

A Figura 23 ilustra o arquivo de configuração do coletor. Nele é possível configurar parâmetros de acesso aos agentes, acesso a base de dados, intervalo de captura em segundos e permitir a visualização de notícias. Os parâmetros são:

```
# Arquivo de configuração do coletord

# Conexão com o agente
snmpComunidade = public
snmpPorta = 161
cltIPsParaCaptura = {"127.0.0.1", "local",
                     "192.168.1.102", "Maq. 1",
                     "192.168.1.100", "Maq. 2",
                     "192.168.1.103", "Maq. 3"}

# Conexão com o servidor da base de dados
bdHost = localhost
bdNome = tcc
bdUsuario = postgres
#bdPassword = postgres
#bdPorta = 5432

# Intervalo de tempo entre as capturas
cltTempoDeCaptura = 300

# 1 exibe noticias
cltExibeNoticia = 0
```

Figura 23 - Arquivo de configuração do coletor.

- *snmpComunidade*: define a comunidade utilizada para acessar os agentes SNMP. A comunidade identifica privilégios de acesso aos objetos da MIB.

- *snmpPorta*: define a porta na qual o SNMP irá responder. Por padrão, a porta do SNMP é a 161.

- *cltIPsParaCaptura*: define a lista de agentes dos quais o coletor deverá requisitar as informações. Para cada agente deve-se especificar o IP seguido do nome da máquina.

- *bdHost*: define o IP da máquina onde o banco de dados está instalado.

- *bdNome*: nome da base de dados que contém as tabelas de processos, temperatura e partição.

- *bdUsuario*: define o usuário utilizado para acessar a base de dados.

- *bdPassword*: define a senha de acesso a base de dados. Caso não possua senha, deve-se comentar o parâmetro.

- *bdPorta*: define a porta na qual o banco de dados responde. Por padrão, o *postgresql* utiliza a porta 5432.

- *cltTempoDeCaptura*: define o tempo de espera em segundos para iniciar uma nova captura das informações dos agentes.

- agtExibeNoticias: utiliza-se este parâmetro para exibir notícias que facilitam o debug do código fonte. O valor 1 habilita a exibição das mensagens e é utilizado durante o desenvolvimento.

Em ambos os arquivos de configuração é possível adicionar comentários ou desabilitar uma opção com o caractere #.

4.6 Console

O console é a interface para o acesso aos dados armazenados na base de dados. Tem por objetivo mostrar para o administrador os dados coletados de todo o parque de máquinas de uma forma rápida, centralizada e acessível.

Não é o objetivo deste trabalho desenvolver um console para acesso ao banco de dados. Porém, existem várias ferramentas para geração de relatórios, que permitem ao administrador acessar o SGBD e extrair as informações necessárias. Um exemplo de ferramenta é o Agata Report que é multi-plataforma e tem suporte ao *postgresql* (Agata, 2007).

4.7 Compilando e executando o SDMR

Como comentado anteriormente, o SDMR é desenvolvido na linguagem C e pode ser compilado com a ferramenta *gcc* (GCC, 2007). Para facilitar a compilação

foi criado um arquivo *Makefile* que encontra-se no Apêndice F. As bibliotecas necessárias para compilar o SDMR são: *libsnmp*, *libconfuse* e *libpq*.

Para compilar o agente e o coletor deve-se digitar *make all*. O utilitário *make* se encarregará de ler o arquivo *Makefile* e executar o *gcc* com os devidos parâmetros (GNU, 2007).

É necessário instalar o *deamon* do SNMP e configurá-lo para aceitar agentes extensíveis. Na distribuição Kubuntu, o arquivo de configuração encontra-se em */etc/snmpd/snmpd.conf*. É necessário editá-lo e descomentar a linha *master agentx*. Após, deve-se reiniciar o *deamon* do SNMP (Netsnmp, 2007).

Deve-se copiar a estrutura da MIB *tcc* para o diretório */usr/share/snmp/mibs* e registra-lá com o comando *echo "mibs +TCC-MIB" >> /usr/share/snmp/snmp.conf.*

Agora é possível rodar o agente para capturar as informações da estação de trabalho. O nome da aplicação é *agented* e deve ser rodada com o usuário *root*.

Na máquina coletora não há a necessidade de ter o NET-SNMP instalado, basta rodar a aplicação *coletord*.

No servidor da base de dados deve-se instalar o SGBD *postgresql* e criar uma base de dados de nome *tcc* (Postgresql, 2007). Após criada a base de dados é necessário criar as tabelas para armazenar as informações extraídas. Os comandos para criar as tabelas encontram-se no Apêndice E.

5 RESULTADOS OBTIDOS

Este capítulo descreve o impacto que o Sistema Distribuído para Monitoração de Recursos - SDMR causará nas estações de trabalho. Além disso, é feita uma avaliação do impacto causado na rede pela transmissão das informações entre o agente e o coletor. Também, avalia-se a quantidade de agentes que um coletor é capaz de suportar. Por fim, avalia-se o SDMR em um ambiente real.

Notou-se que ao repetir as avaliações a seguir, os resultados obtidos eram praticamente iguais. Isso deve-se ao fato de que o ambiente controlado teve pouca variação na quantidade de processos.

5.1 Impactos causados pelo agente na estação de trabalho

Esta avaliação tem por objetivo identificar o uso dos recursos de processador e memória pelo agente em uma estação de trabalho. Para isso, foram executados

quatro testes em diferentes cenários. A Tabela 1 mostra o intervalo de captura em segundos para processos, temperaturas e partições. Para cada cenário dobrou-se o intervalo de captura das informações.

Tabela 1 - Especificação do intervalo de captura em segundos para cada cenário.

Cenário	Processos (s)	Temperatura (s)	Partições (s)
A	2	60	360
B	4	120	720
C	8	240	1440
D	16	480	2880

O tempo de duração do teste para cada cenário foi de 1 hora, ou seja, o agente coletou as informações em uma estação de trabalho e após 1 hora o coletor requisitou as informações.

Para obter os resultados, o agente rodou em uma máquina com processador Intel® Core™ 2 Duo de 1,66 Ghz, com distribuição Kubuntu e interface gráfica. Para garantir que o agente execute por 1 hora, desenvolveu-se um *script* em *shell* que é executado logo após o agente. Como ilustrado na Figura 24, o *script* recebe como parâmetro ($1) o PID do processo agente e executa quatro comandos. No primeiro comando é exibido o estado da memória inicial do agente. O segundo comando deixa o *script* esperando por 3.600 segundos enquanto o agente extrai as informações. O terceiro comando, exibe novamente o estado da memória do agente, só que desta vez, 1 hora após o início do *script*. Por fim, é executado o coletor para requisitar as informações do agente.

```
#!/bin/bash
cat /proc/$1/statm && sleep 3600 && cat /proc/$1/statm && ./coletord
```

Figura 24 - *Script*.

A informação de memória é exibida pelo *script*. Esta informação encontra-se na unidade de blocos e é convertida para KB multiplicando-se os blocos por 4. O processo de conversão foi explicado na Seção 4.1. Já, a informação de processamento é retirada da base de dados do SDMR, onde é encontrada em percentual de ocupação.

O teste foi executado em uma estação de trabalho em produção. Assim, para cada teste executado houve uma pequena variação na quantidade de informações capturadas. A Tabela 2 exibe a quantidade de nodos da seqüência mantida pela MIB para os processos, temperatura e partições. Para obter a quantidade de nodos capturadas basta habilitar a exibição de notícias no arquivo de configuração do coletor.

Tabela 2 - Total de nodos capturadas por cada teste.

Cenário	Processos	Temperatura	Partições
A	13.967	61	132
B	8.977	31	72
C	5.522	16	36
D	3.459	8	24

A Tabela 3 mostra o resultado obtido em cada cenário avaliado. A segunda coluna mostra uso do processador em percentual. A terceira coluna exibe a quantidade de memória em KB ocupada pelo agente e pelos dados capturados. Por

fim, a quarta coluna mostra somente a quantidade de memória ocupada pelos dados capturados. Esta última informação foi obtida através da subtração da memória ocupada pelo agente antes da captura com a memória ocupada pelo agente antes da requisição do coletor. A memória ocupada pelo agente antes de coletar as informações é de 3.072 KB.

Tabela 3 - Uso dos recursos de CPU e memória pelo agente.

Cenário	Ocupação da CPU (%)	Ocup. da Memória pelo Agente (KB)	Ocup. da Memória pelos Dados (KB)
A	1,47	9.268	6.196
B	0,82	7.044	3.972
C	0,4	5.548	2.457
D	0,2	4.632	1.560

A Figura 25 ilustra o gráfico de relação entre ocupação da CPU (%) e os cenários avaliados. Desconsiderando a pequena variação sofrida na captura das informações, pode-se dizer que o gráfico possui um comportamento linear. Assim, deduz-se que ao dobrar o intervalo de captura o uso do processador cairá pela metade.

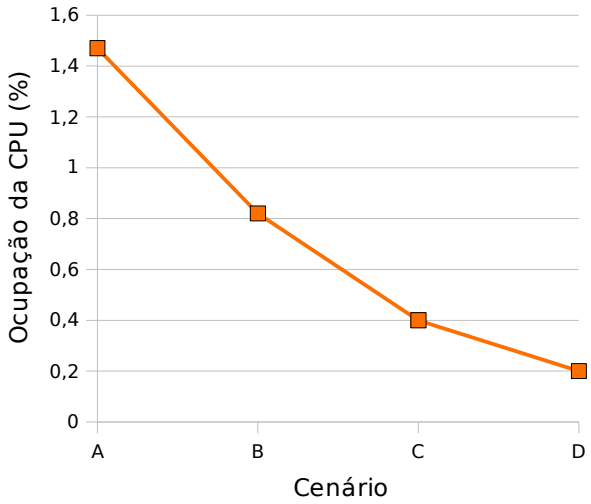


Figura 25 - Comparação do uso do processador de um agente em diferentes cenários.

A Figura 26 ilustra o gráfico de relação entre ocupação de Memória (KB) e os cenários avaliados. O gráfico mostra a memória utilizada pelos dados e a memória total utilizada pelo agente. Desconsiderando a pequena variação sofrida na captura das informações, pode-se considerar que a ocupação de memória pelos dados é linear. Assim, ao dobrar o intervalo de captura a ocupação de memória cai pela metade.

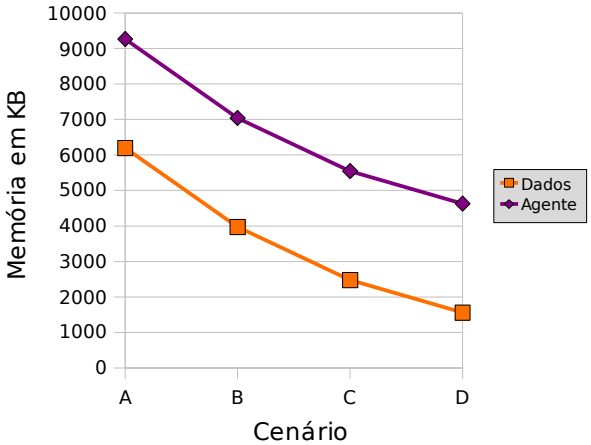


Figura 26 - Comparação do uso de memória de um agente em diferentes cenários.

5.2 Impacto causado na rede pela transmissão das informações entre o agente e o coletor

Esta avaliação monitora a quantidade e tamanho total dos pacotes que trafegam entre um agente e um coletor. Utiliza-se o mesmo cenário definido na Tabela 1 e avalia-se o grupo de cenários nos intervalos de coleta 30, 20 e 7,5 minutos. Com isso, obtém-se um total de 12 testes em diferentes cenário para diferentes tempos.

Para obter os resultados, o agente e o coletor foram executados em uma máquina com processador Intel® Core™ 2 Duo de 1,66 Ghz, com distribuição Kubuntu e interface gráfica. Para garantir que o agente seja executado nos tempos definidos acima, desenvolveu-se um *script* em *shell* que é executado logo após o agente. Como ilustrado na Figura 27, o *script* espera um determinado tempo em segundos e executa o coletor para requisitar as informações do agente.

```
#!/bin/bash

sleep 450 && ./coletord
```

Figura 27 - *Script* 2.

As informações sobre quantidade e tamanho total dos pacotes que trafegam entre o agente e o coletor foram obtidas com o auxílio da ferramenta *Wireshark*. Esta ferramenta analisa o tráfego da rede e possibilita a contabilização de pacotes através de filtros por protocolos (Wireshark, 2007). Assim, monitorou-se a rede capturando todos os pacotes do protocolo SNMP que trafegaram entre o agente e o coletor. Com isso é possível obter a quantidade de pacotes e o tamanho total em KB da transmissão.

Assim como o teste anterior, este foi executado em uma estação de trabalho em produção. Com isso, entre os testes houve uma variação na quantidade de informações capturadas. Esta variação é levada em consideração para concluir o teste.

A Tabela 4 exibe o resultado obtido para cada cenário avaliado. A segunda coluna exibe o tempo em que o agente executou antes que coletor requisitasse as informações. A terceira coluna exibe o número de pacotes relacionados ao protocolo SNMP que trafegaram na rede. A quarta coluna exibe o tamanho total em KB da transmissão entre o agente e o coletor. A quinta coluna exibe o número de nodos recebidos pelo coletor. Por fim, é feita a relação entre o tamanho total em KB com o número de linhas recebidas.

Tabela 4 - Uso dos recursos da rede pela transmissão de informações entre agente e coletor

Cenário	Intervalo de coleta (min)	Pacotes	Tam. total (KB)	Nº de nodos	KB/nodo
A	30	63.217	5.904,71	4.551	1,3
B	30	40.000	3.735,85	2.871	1,3
C	30	28.636	2.675,16	2.053	1,3
D	30	21.874	2.042,88	1.566	1,3
A	15	37.746	3.523,77	2.710	1,3
B	15	22.366	2.088,54	1.616	1,29
C	15	15.280	1.426,89	1.104	1,29
D	15	11.871	1.107.84	867	1,28
A	7,5	17.478	1.630,28	1.256	1,3
B	7,5	12.228	1.140.92	877	1,3
C	7,5	7.008	653,22	503	1,3
D	7,5	6.456	601,59	463	1,3

A Figura 28 expressa o tráfego real entre uma estação de trabalho em produção e o coletor. Nota-se que há uma variação entre os testes e não torna-se possível identificar se o comportamento é linear ou exponencial. Porém, é possível avaliar que ao dobrar o intervalo de captura das informações há uma queda significativa no tráfego da rede.

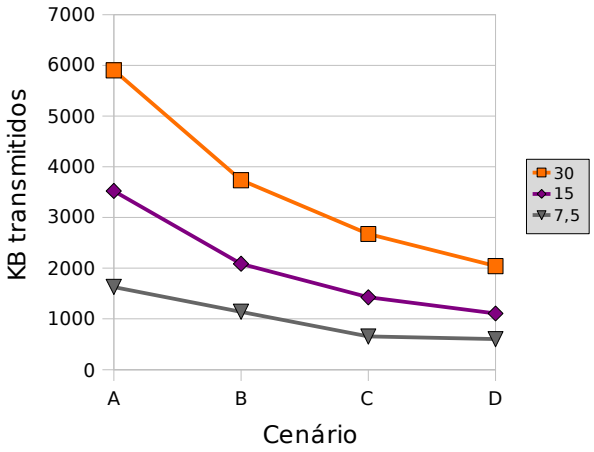


Figura 28 - Comparação do tráfego da rede entre os intervalos de captura e cenário.

Nota-se, na tabela 4, que em qualquer cenário e duração do teste a transmissão de um nodo ocupa o mesmo tráfego de rede. Assim, pode-se deduzir que para o tráfego gerado pelo SDMR é linear. Ou seja, o cenário 4 transmite em um intervalo de coleta de 7,5 minutos um tráfego de 463 KB. Então, ao dobrar o intervalo de captura das informações, o mesmo tende a utilizar um tráfego de 231,5 KB em um intervalo de coleta de 7,5 minutos.

5.3 Quantidade de agentes por coletor

Esta avaliação tem por objetivo identificar a quantidade teórica máxima de agentes por coletor. Para isso, avalia-se o uso do processador, memória e o tempo necessário para coletar/processar as informações capturadas pelos agentes nas estações de trabalho. Criaram-se 12 novos cenários baseados nos cenários

anteriores, mas acrescidos de 3 diferentes números de agentes. Os novos cenários são ilustrados pela Tabela 5.

Tabela 5 - Especificação do número de agentes e intervalo de captura em segundos para cada cenário

Cenário	Nº de agentes	Processos (s)	Temperatura (s)	Partições (s)
1.A	10	2	60	360
1.B	10	4	120	720
1.C	10	8	240	1440
1.D	10	16	480	2880
2.A	20	2	60	360
2.B	20	4	120	720
2.C	20	8	240	1440
2.D	20	16	480	2880
3.A	30	2	60	360
3.B	30	4	120	720
3.C	30	8	240	1440
3.D	30	16	480	2880

Para o teste utilizou-se um ambiente controlado, ou seja, não havia produção nas estações de trabalho. Porém, para que houvesse um processamento mínimo deixou-se aberto nas estações de trabalho um navegdor Web acessando um site em *flash*. Assim, todas as máquinas tiveram um comportamento parecido mas não igual.

O ambiente é composto por no máximo 30 agentes, um coletor e um servidor de base de dados. Os agentes e o coletor foram executados em uma máquina Intel® Pentium® 4 CPU 3.00GHz, com Kubuntu e interface gráfica. Os cenários acima formam testados nos intervalos de coleta 30, 15 e 7,5 minutos, obtendo-se um total

de 46 testes. Para garantir que o agente seja executado nos tempos definidos acima, desenvolveu-se um *script* em *shell* que é executado logo após o início dos agentes. Como ilustrado na Figura 29, o *script* espera um determinado tempo em segundos e executa o coletor para requisitar as informações dos agentes.

```
#!/bin/bash

sleep 1800  && time ./coletord
```

Figura 29 - Script 3.

A informação de uso do processador é retirada da base de dados do SDMR, onde é encontrada em percentual de ocupação. Para capturar a informação de tempo de execução utiliza-se a ferramenta *time*. Por fim, obtém-se a informação de memória alterando o código fonte do coletor e adicionando o código ilustrado pela Figura 30 após as coletas de processos, temperaturas e partições. Isso deve-se ao fato de que o coletor libera a memória utilizada por cada captura. Esta informação é exibida em tela e para as estatísticas é pego o maior valor de memória alocada.

```
sprintf(mem, "cat /proc/%d/statm", getpid());
system(mem);
```

Figura 30 - Código fonte em C para visualizar os blocos de memória utilizados pelo processo.

As Tabelas 6, 7 e 8 exibem o resultado obtido em cada cenário nos respectivos tempos. A segunda coluna exibe o valor em percentual da ocupação do processador diluído no tempo do duração do teste. Esta informação foi obtida na base de dados do SDMR. A terceira coluna exibe o percentual de ocupação do processador no tempo da coleta. A quarta coluna exibe o tempo que o coletor levou para requisitar, receber, processar e armazenar as informações no banco de dados. Por fim, a quinta coluna é a quantidade teórica máxima de agentes suportado pelo coletor no ambiente de teste.

Tabela 6 - Utilização dos recursos pelo coletor no intervalo de coleta 30 minutos.

Cenário	% CPU (Duração)	% CPU (Coleta)	Memória (KB)	Tempo de coleta (s)	Nº máx. de agentes
1.A	0,58	6,07	4.220	172	104
1.B	0,38	7,05	3.960	97	185
1.C	0,23	7,14	3.806	58	310
1.D	0,16	7,58	3.736	38	473
2.A	1,16	5,6	4.236	373	96
2.B	0,68	6,18	3.956	198	181
2.C	0,45	7,17	3.860	113	318
2.D	0,31	8,21	3.740	68	529
3.A	1,4	4,5	4.280	560	96
3.B	1,03	6,12	3.968	303	178
3.C	0,63	6,75	3.816	168	321
3.D	0,46	8,12	3.740	102	529

Tabela 7 - Utilização dos recursos pelo coletor no intervalo de coleta 15 minutos.

Cenário	% CPU (Duração)	% CPU (Coleta)	Memória (KB)	Tempo de coleta (s)	Nº máx. de agentes
1.A	0,59	6,17	3.900	86	104
1.B	0,36	6,48	3.828	50	180
1.C	0,27	8,1	3.820	30	300
1.D	0,21	8,59	3.684	22	409
2.A	1,07	5,26	3.932	183	98
2.B	0,73	6,5	3.856	101	178
2.C	0,48	7,45	3.720	58	310
2.D	0,36	8,1	3.688	40	450
3.A	1,77	5,22	3.992	305	88
3.B	1,07	6,21	3.884	155	174
3.C	0,66	6,83	3.844	87	310
3.D	0,52	8,07	3640	58	465

Tabela 8 - Utilização dos recursos pelo coletor no intervalo de coleta 7,5 minutos.

Cenário	% CPU (Duração)	% CPU (Coleta)	Memória (KB)	Tempo de coleta (s)	Nº máx. de agentes
1.A	0,6	5,87	3.760	46	97
1.B	0,45	7,5	3.704	27	166
1.C	0,29	8,16	3.660	16	281
1.D	0,23	8,63	3.612	12	375
2.A	1,19	5,41	3.776	99	90
2.B	0,9	7,5	3.712	54	166
2.C	0,59	8,3	3.660	32	281
2.D	0,54	10,13	3.620	24	375
3.A	1,93	5,53	3.784	157	85
3.B	1,27	6,72	3.724	85	158
3.C	0,9	8,27	3.660	49	275
3.D	0,74	9,79	3.632	34	397

Para calcular o percentual de ocupação do processador no tempo da coleta foi utilizada a equação $ptc = \frac{(ptd * tdt)}{tc}$. Onde *ptc* é o percentual no tempo da coleta, *ptd* é o percentual do tempo de duração do teste, *tdt* é o tempo de duração do teste em segundos e *tc* é o tempo de coleta em segundos.

A seguir, compara-se o uso do processamento, memória e o intervalo de coleta das informações. Após comenta-se sobre o número máximo de agentes por coletor.

Na Figura 31 verifica-se que ao dobrar o intervalo de captura das informações nos agentes o uso do processador cai consideravelmente. Nota-se também que o uso do processador em relação aos intervalos de coleta é praticamente o mesmo. Este comportamento está relacionado com o método de coleta seqüencial executado pelo coletor. Assim, ao aumentar o intervalo de coleta do teste o uso do processador será sempre o mesmo. Porém, levará mais tempo para requisitar as informações de todos os agentes. Como isso, deduz-se que o limite de agentes por coletor está relacionado em requisitar todas as informações no intervalo de tempo da coleta.

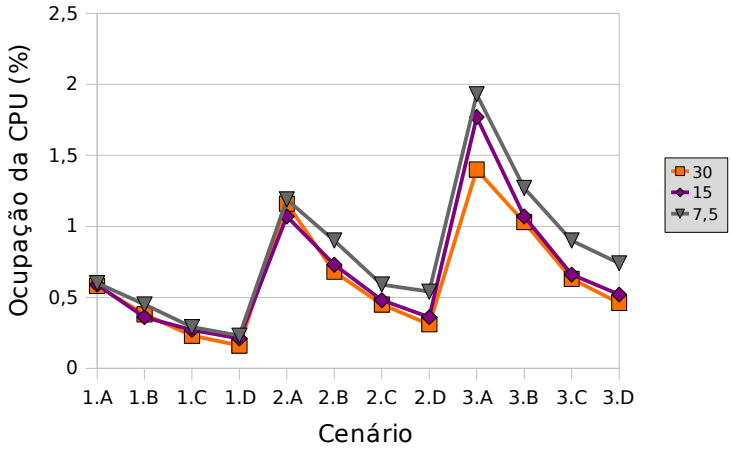


Figura 31 - Comparação do uso do processador pelo coletor entre os intervalos de coleta.

Na Figura 32 visualiza-se que ao aumentar o intervalo de coleta do teste o tempo que o coletor leva para coletar, processar e armazenar as informações também aumenta. Com isso, confirma-se que a restrição de número de agentes está relacionada ao intervalo de coleta e não ao processamento.

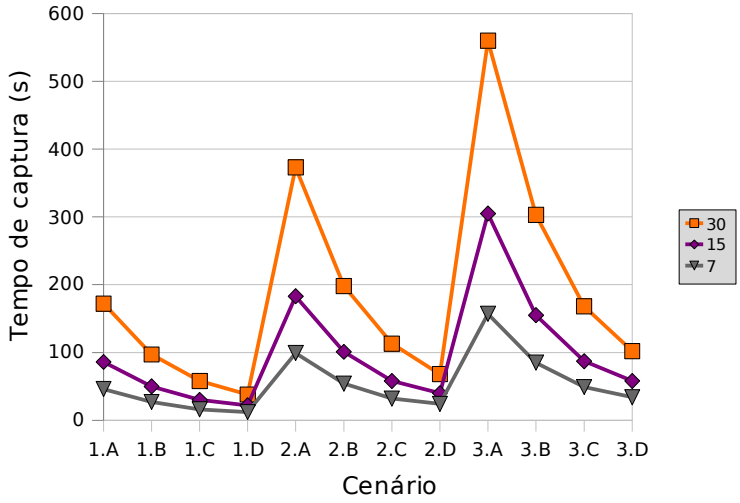


Figura 32 - Comparação do intervalo de captura pelo coletor entre os intervalos de coleta.

Na Figura 33 visualiza-se que ao dobrar o intervalo de captura das informações nos agentes o uso da memória na máquina coletora cai consideravelmente. Nota-se também que ao adicionar mais agentes (1.A, 2.A, 3.A) a ocupação de memória se mantém. Este comportamento está relacionado com o método de coleta seqüencial executado pelo coletor. Desta forma, o coletor aloca a memória para as informações de um agente e após o procedimento de armazenamento libera a memória alocada.

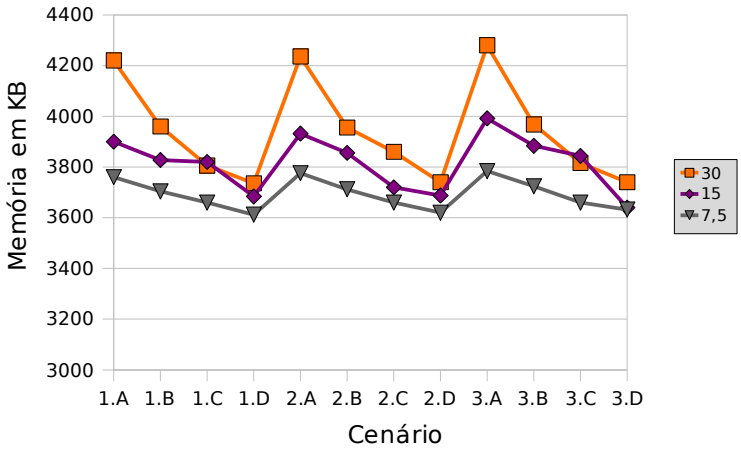


Figura 33 - Comparação do uso da memória pelo coletor entre os intervalos de coleta.

Como visto anteriormente, o número máximo de coletores está relacionado ao tempo de duração da coleta. Assim, a estimativa de agentes exibida na sexta coluna das tabelas 6, 7 e 8 foram calculadas utilizando a equação $e=\frac{(tdt*na)}{tc}$. Onde *e* é a estimativa de agentes por coletor, *tdt* é o tempo de duração do teste em segundos, *na* é o número de agentes utilizados no teste e *tc* é o tempo de coleta em segundos.

Nota-se na Figura 34 que ao adicionar mais agentes (1.A, 2.A, 3.A) para cada intervalo de captura a estimativa de agentes manteve-se praticamente a mesma. Ou seja, para o intervalo de coleta 30 minutos a estimativa para 10, 20 e 30 agentes manteve-se entre 96 e 104 agentes por coletor. Visualiza-se também que ao dobrar o intervalo de captura das informações nos agentes a estimativa de máquinas por coletor sobe consideravelmente.

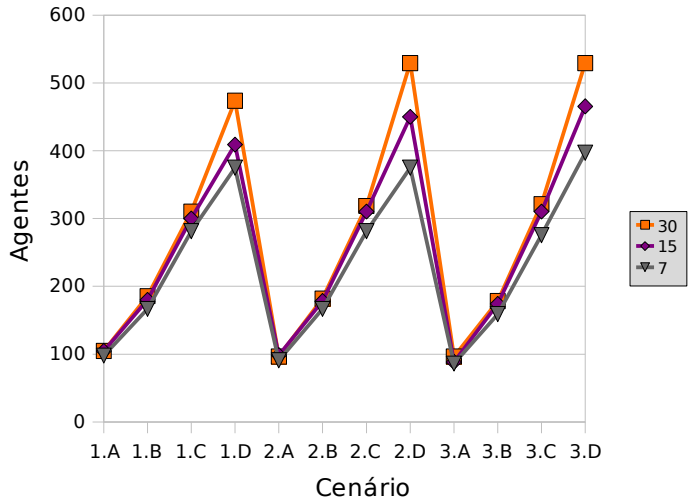


Figura 34 - Comparação da estimativa de agentes entre os intervalos de coleta e cenários.

5.4 Avaliação em ambiente de produção

Esta avaliação tem por objetivo verificar o comportamento do SDMR em um ambiente real. O ambiente é composto por um servidor de base de dados, um coletor e sete agentes. A avaliação foi executada em um tempo total de 2 horas e 30 minutos e foram executadas 15 coletas.

Os agentes foram configurados para capturar as informações locais em um intervalo de 32 segundos para os processos, 480 segundos para a temperatura e 2880 segundos para as partições. O coletor foi configurado para coletar as informações a cada 10 minutos.

A Figura 35 ilustra a estrutura montada para a avaliação no ambiente real. Visualiza-se sete máquinas, onde seis possuem somente o agente e a M7 possui o agente, o coletor e a base de dados.

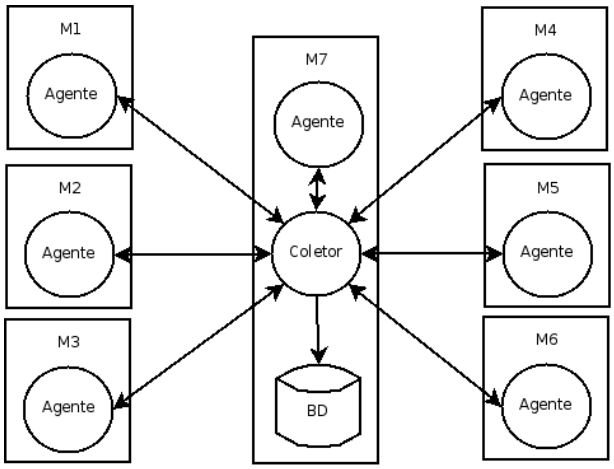


Figura 35 - Estrutura montada para a avaliação no ambiente real.

A Tabela 9 exibe o modelo do processador das máquinas utilizadas no ambiente.

Tabela 9 - Modelo do processador das máquinas utilizadas no ambiente.

Máquina	Modelo do processador
M1	Genuine Intel® CPU T2300 1.66GHz
M2	Intel® Core™ 2 CPU T5200 1.60GHz
M3	Intel® Core™ 2 CPU T5500 1.66GHz
M4	Intel® Pentium® 4 CPU 2.40GHz
M5	Intel® Celeron® CPU 2.66GHz
M6	Intel® Celeron® CPU 2.53GHz
M7	Intel® Core™ 2 CPU T5500 1.66GHz

Para obter os resultados da avaliação, alterou-se o código fonte do agente e coletor, adicionando o código ilustrado pela Figura 30. Assim, obtém-se a informação

de uso de memória, e o tempo de execução da cada coleta consumidos pelo coletor, bem como o uso de memória do agente. O uso de processador pelo agente é retirada da própria base de dados do SDMR.

A Tabela 10 exibe os resultados do consumo de processador pelo agente em cada máquina. Nota-se que o processamento do agente mantem-se o mesmo para cada máquina durante as 15 coletas.

Tabela 10 - Ocupação do processador pelo processo agente.

Coleta	M1 (%)	M2 (%)	M3 (%)	M4 (%)	M5 (%)	M6 (%)	M7 (%)
1	0,12	0,12	0,09	0,07	0,09	0,13	0,12
2	0,11	0,13	0,14	0,1	0,1	0,25	0,14
3	0,13	0,16	0,16	0,1	0,22	0,27	0,16
4	0,12	0,13	0,15	0,1	0,3	0,27	0,14
5	0,12	0,17	0,14	0,1	0,15	0,3	0,13
6	0,12	0,14	0,14	0,09	0,15	0,3	0,15
7	0,12	0,18	0,14	0,11	0,21	0,28	0,14
8	0,12	0,16	0,13	0,11	0,2	0,38	0,14
9	0,12	0,15	0,13	0,1	0,18	0,2	0,16
10	0,11	0,17	0,16	0,1	0,16	0,24	0,13
11	0,12	0,17	0,15	0,11	0,2	0,29	0,15
12	0,12	0,16	0,14	0,12	0,18	0,3	0,15
13	0,12	0,13	0,12	0,11	0,17	0,24	0,14
14	0,12	0,15	0,13	0,09	0,24	0,38	0,14
15	0,12	0,13	0,14	0,09	0,19	0,16	0,12

A Tabela 11 exibe os resultados do consumo da memória em KB pelo processo agente em cada máquina. A informação da memória é capturada um momento antes do coletor requisitar a memória. Nota-se que a ocupação de memória, apesar de aumentar em poucos KB, manteve-se praticamente a mesma para cada máquina durante as 15 coletas.

Tabela 11 - Ocupação de memória em KB pelo processo agente.

Coleta	M1	M2	M3	M4	M5	M6	M7
1	3.404	3.272	3.468	3.324	3.824	3.284	3.364
2	3.408	3.432	3.468	3.324	4.016	3.344	3.364
3	3.408	3.408	3.468	3.324	4.040	3.360	3.364
4	3.408	3.408	3.468	3.324	3.888	3.368	3.376
5	3.408	3.428	3.484	3.332	3.888	3.392	3.376
6	3.412	3.428	3.484	3.332	3.880	3.400	3.376
7	3.412	3.428	3.484	3.332	3.880	3.404	3.376
8	3.412	3.432	3.488	3.332	3.792	3.420	3.380
9	3.412	3.432	3.488	3.332	3.776	3.420	3.380
10	3.412	3.432	3.488	3.336	3.808	3.424	3.380
11	3.416	3.432	3.488	3.336	3.808	3.424	3.380
12	3.416	3.432	3.488	3.336	3.824	3.424	3.380
13	3.416	3.436	3.488	3.336	3.784	3.424	3.384
14	3.416	3.436	3.488	3.336	3.784	3.424	3.384
15	3.416	3.436	3.492	3.340	3.784	3.500	3.384

A Tabela 12 mostra a utilização dos recursos pelo coletor em cada coleta. A segunda coluna exibe o valor em percentual da ocupação do processador diluído no tempo do duração do teste. Esta informação foi obtida na base de dados do SDMR. A terceira coluna exibe o percentual de ocupação do processador no tempo da

coleta. A quarta coluna exibe o tempo que o coletor levou para requisitar, receber, processar e armazenar as informações no banco de dados. Por fim, a quinta coluna é a estimativa do número máximo de agentes suportado pelo coletor.

Tabela 12 - Utilização dos recursos pelo coletor no intervalo de coleta 10 minutos.

Coleta	% CPU (Duração)	% CPU (Coleta)	Memória (KB)	Tempo de coleta (s)	Nº máx. de agentes
1	0,22	3,38	4.064	39	107
2	0,43	6,29	4.068	41	102
3	0,17	3	4.084	34	123
4	0,41	6,83	4.096	36	116
5	0,38	6,91	4.088	33	127
6	0,37	5,05	4.080	44	95
7	0,21	3,32	4.088	38	110
8	0,4	6,49	4.088	37	113
9	0,37	6	4.084	37	113
10	0,41	6,15	4.072	40	105
11	0,2	3,16	4.088	38	110
12	0,38	6,33	4.100	36	116
13	0,35	5,83	4.092	36	116
14	0,15	3,33	4.088	27	155
15	0,37	6,94	4.084	32	131

Nota-se que a ocupação de memória comportou-se de forma parecida para as diferentes coletas. Este comportamento está relacionado com o método de coleta seqüencial executado pelo coletor. Desta forma, o coletor aloca a memória para as informações de um agente e após o procedimento de armazenamento libera a memória alocada.

Na Figura 36 nota-se que o coletor variou a ocupação do processador, porém na maioria das coletas manteve-se concentrada em uma faixa. O processamento varia de acordo com a quantidade de informações capturadas pelo agente.

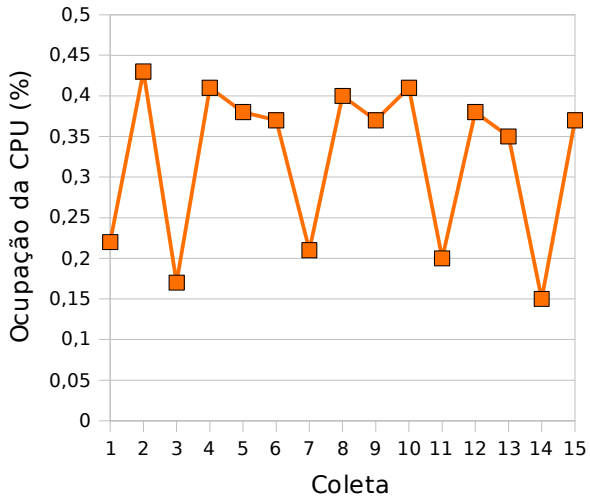


Figura 36 - Comparação do uso do processador pelo coletor em diferentes coletas.

O número máximo de agentes por coletor foi calculado da mesma forma que na seção 5.3. Nota-se que para cada coleta o número máximo de agentes por coletor varia. Esta variação está relacionada com a quantidade de informações capturadas pelos agentes. Para este teste o coletor suportaria 95 agentes, porém trabalharia no limite máximo.

Na Tabela 13 visualiza-se o uso da rede pela transmissão das informações entre seis agentes e o coletor. O agente da máquina M7 não influencia na rede pois está instalado na mesma máquina que o coletor. As informações de pacotes e KB transmitidos foram obtidas a cada coleta e com o auxílio da ferramenta *Wireshark.*

Tabela 13 - Uso dos recursos da rede pela transmissão das informações entre seis agentes e o coletor.

Coleta	Pacotes	KB transmitidos
1	2.7274	2.538,31
2	2.6944	2.507,07
3	2.3487	2.184,63
4	2.5203	2.344,48
5	2.0566	1.910,96
6	2.8060	2.611,71
7	2.7252	2.535,91
8	2.3630	2.197,63
9	2.0826	1.935,87
10	2.7336	2.543,57
11	2.5404	2.362,98
12	2.3400	2.175,65
13	2.5296	2.353,25
14	2.2968	2.135,73
15	2.3010	2.139,70

A Figura 37 ilustra o tráfego causado pela transmissão das informações entre seis agentes e o coletor em cada intervalo de coleta. Nota-se que a média de transmissão das informações dos seis agentes a cada 10 minutos é de 2.298,5 KB.

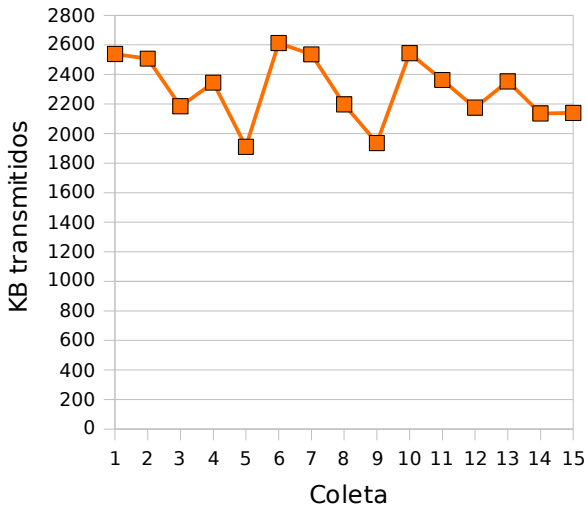


Figura 37 - Tráfego da rede causado pela transmissão das informações entre seis agentes e o coletor.

5.5 Análise geral dos resultados obtidos

Analisando os resultados obtidos verifica-se que o uso SDMR torna-se aceitável, visto que o agente e o coletor utilizaram poucos recursos das máquinas onde estão hospedados. O tráfego das informações se torna adequado, já que o SDMR utiliza a estrutura de uma rede local e é uma pequena fração da banda total.

Também verificou-se que ao aumentar o intervalo de captura do agente os impactos causados no ambiente são bem menores. Com isso, é possível chegar a uma configuração ideal para cada ambiente.

6 CONCLUSÃO

Este trabalho apresentou a implementação de um sistema distribuído para monitorar o uso dos recursos de hardware e software em estações de trabalho GNU/Linux. Inicialmente foram descritos conceitos sobre o monitoramento de recursos de hardware e softwares no sistema operacional GNU/Linux. Também, foi feita uma análise das soluções existentes e identificou-se a necessidade de um sistema para tal propósito. Em seguida, foi descrita a implementação identificando a estrutura do sistema e a maneira como as informações são obtidas. Por fim, foram feitas avaliações sobre o impacto do sistema no ambiente e chegou-se à conclusão que o seu uso é viável.

Para a implementação do sistema foi desenvolvida uma aplicação agente que executa nas estações que se deseja monitorar. O agente utiliza chamadas de sistema e consulta o diretório */proc* para a captura das informações. Para a coleta e armazenamento das informações foi desenvolvida uma aplicação coletora, chamada

de coletor. As informações são armazenadas em um banco de dados para futuras consultas.

Este trabalho contribuiu com uma importante ferramenta para monitorar o uso dos recursos de hardware e software em estações de trabalho GNU/Linux que, até então, não existia. Com esta ferramenta o gerente de TI tem em mãos informações de todas as máquinas GNU/Linux da organização. Estas informações podem auxiliar na tomada de decisão, bem como, na realocação de recursos, atualização de maquinário e monitoramento dos processos.

Como trabalhos futuros pretende-se a implementação das seguintes funcionalidades:

- Desenvolvimento da aplicação console: desenvolver uma aplicação específica para a exibição do dados coletados. Com isso, será possível visualizar as informações de forma amigável com o auxilio de gráficos.

- Possibilitar a utilização de *threads* no coletor: atualmente o coletor utiliza a requisição seqüencial para requisitar informações dos agentes. Com isso, o limite máximo de agentes está relacionado com o tempo de coleta/processamento das informações. Assim, é interessante que o coletor possa requisitar as informações de um agente enquanto processa as informações de outro. Desta forma, o limite de agentes estará relacionado à limitação do hardware e não mais a um intervalo de tempo.

- Detecção automática de novos agentes: atualmente o administrador deve informar para o coletor quais são os agentes dos quais ele deverá coletar as informações. Assim, é interessante que o coletor seja capaz de detectar um novo agente na rede e passar a coletar as informações do mesmo sem a necessidade de intervenção do administrador.

- Captura de novas informações: implementar no agente a captura de novas informações da estação de trabalho.

- Suporte para outros sistemas operacionais: para que o SDMR suporte a captura de informações de outros sistemas operacionais será necessária somente a reescrita do agente.

REFERÊNCIAS

AGATA. **Home.** Disponível em: <http://agata.solis.coop.br> Acesso em: 25 nov. 2007.

BOVET, Daniel P.; CESATI, Marco. **Understanding the Linux Kernel**. 3 ed. Beijing: O'Reilly, 2006.

CACIC. **Downloads.** Disponível em: <http://guialivre.governoeletronico.gov.br/cacic /sisp2/downloads/donwloads.htm> Acesso em: 20 mai. 2007.

COMER, Douglas E.. **Interligação em rede com TCP/IP: principios, protocolos e arquitetura**. Rio de Janeiro: Campus, 1998.

CORBET, Jonathan; RUBINI, Alessandro; KROAH-HARTMAN, Greg. **Linux: device drivers**. 3 ed. Beijing: O Reilly, 2005.

ELMASRI, Ramez; NAVATHE, Shamkant B.. **Sistemas de banco de dados**. São Paulo: Pearson Addison Wesley, 2005.

GCC. **GCC, the GNU Compiler Collection**. Disponível em: <http://gcc.gnu.org> Acesso em: 02 ago. 2007.

GNU. **Using sysconf**. Disponível em: <http://www.gnu.org/software/libc/manual /html_node/Sysconf.html> Acesso em: 21 ago. 2007.

HTOP. **Home**. Disponível em: <http://htop.sourceforge.net> Acesso em: 05 nov. 2007.

HYPERIC. **Home.** Disponível em: <http://www.hyperic.com> Acesso em: 23 mai. 2007.

IVIRTUA. **Home.** Disponível em: <http://www.ivirtua.com.br> Acesso em: 19 mai. 2007.

LIBCONFUSE. **Home**. Disponível em: <http://www.nongnu.org/confuse> Acesso em: 2 out. 2007.

LINUXINSIGHT. **The /proc filesystem documentation.** Disponível em: <http://www.linuxinsight.com/proc_filesystem.html> Acesso em: 15 jun. 2007.

LOVE, Robert. Linux Kernel Development. 2 ed. Indianapolis: Novel, 2005.

MITCHELL, Mark; OLDHAM, Jeffrey; SAMUEL, Alex. **Advanced Linux programming**. 1 ed. New Riders Publishing, 2001. Diponível em: <http://www.advancedlinuxprogramming.com> Acesso em: 01 ago. 2007.

MIZRAHI, Victorine Viviane. **Treinamento em linguagem C**. São Paulo: Makron Books, 1990.

MORAES, Gleicon da Silveira. **Programação avançada em Linux**. 1 ed. São Paulo: Novatec, 2005.

NETEYE. **Home**. Disponível em: <http://www.neteye.com.br> Acesso em: 09 mai. 2007.

NETSNMP. **Home**. Disponível em: <http://net-snmp.sourceforge.net> Acesso em: 04 set. 2007.

OLIVEIRA, Rômulo Silva de; CARISSIMI, Alexandre da Silva; TOSCANI, Simão Sirineo. **Sistemas operacionais.** 2 ed. Porto Alegre: Instituto de Informática da UFRGS, 2001.

PISCITELLO, D. M. e CHAPIN, A. L.. **Open systems networking: TCP/IP and OSI**. MA: Addison-Wesley, 1993.

POSTGRESQL. **Home**. Disponível em: <http://www.postgresql.org> Acesso em: 15 out. 2007.

PUPPET. **Home.** Disponível em: <http://www.reductivelabs.com> Acesso em: 20 mai. 2007.

RODRIGUEZ, Claudia Salzberg; FISCHER, Gordon; SMOLSKI, Steven. **The Linux Kernel primer: a top-down approach for x86 and PowerPC architectures**. Boston: Prentice Hall PTR, 2006.

TANENBAUM, Andrew S.. **Redes de computadores**. 4 ed. Rio de Janeiro: Campus, 1997.

TENENBAUM, Aaron M.; LANGSAM, Yedidyah; AUGENSTEIN, Moshe J.. **Estrutura de dados usando C**. São Paulo: Makron Books, 1995.

WIRESHARK, **Home**. Disponível em: <http://www.wireshark.org/> Acesso em: 20 nov. 2007.

ZENOSS. **Home.** Disponível em: <http://www.zenoss.com> Acesso em: 24 mai. 2007.

APÊNDICES

LISTA DE APÊNDICES

APÊNDICE A Estrutura da MIB

Abaixo segue o código fonte completo da MIB, denominada TCC-MIB.txt. Foi adicionada na árvore com o identificador 1000 e abaixo do objeto netSnmp. Assim, para acessar o objeto *tcc,* utiliza-se o endereço 3.6.1.4.1.8072.1000.

```
1   TCC-MIB DEFINITIONS ::= BEGIN
2
3   IMPORTS
4       MODULE-IDENTITY, OBJECT-TYPE, Integer32,
5       NOTIFICATION-TYPE                       FROM SNMPv2-SMI
6       SnmpAdminString                         FROM SNMP-FRAMEWORK-MIB
7       netSnmp                                 FROM NET-SNMP-MIB
8       RowStatus, StorageType                  FROM SNMPv2-TC
9       InetAddressType, InetAddress            FROM INET-ADDRESS-MIB
10  ;
11
12  tcc MODULE-IDENTITY
13      LAST-UPDATED "200710230000Z"
14      ORGANIZATION ""
15      CONTACT-INFO
16        ""
17      DESCRIPTION
18       "MIB implementada para armazenar temporariamente dados da máquina local"
19      ::= { netSnmp 1000 }
20
21
22  tccTabela        OBJECT IDENTIFIER ::= { tcc 1 }
23
24
25  tccProcessTable OBJECT-TYPE
26      SYNTAX      SEQUENCE OF TccEntradaParaProcessos
27      MAX-ACCESS  not-accessible
28      STATUS      current
29      DESCRIPTION
30       "Armazena temporariamente informações sobre os processos
31           que estão rodando na máquina local."
32      ::= { tccTabela 1 }
33
34  tccEntradaParaProcessos OBJECT-TYPE
35      SYNTAX      TccEntradaParaProcessos
36      MAX-ACCESS  not-accessible
37      STATUS      current
38      DESCRIPTION
39       "Identifica uma linha da tabela"
40      INDEX   { pIDDaLinha }
41      ::= {tccProcessTable 1 }
42
43  TccEntradaParaProcessos ::= SEQUENCE {
44       pIDDaLinha          Integer32,
45       pID         Integer32,
46       pDataHoraDaColeta OCTET STRING,
47       pNome       OCTET STRING,
48       pUsuario    OCTET STRING,
```

```
49       pUtilizacaoDaCPU         OCTET STRING,
50       pUtilizacaoDaMemoria     OCTET STRING
51  }
52
53  pIDDaLinha OBJECT-TYPE
54      SYNTAX      Integer32 (0..2147483647)
55      MAX-ACCESS  read-only
56      STATUS      current
57      DESCRIPTION
58       "Número da linha. Este valor é reiniciado pelo agente
59           conforme as requisiçoes de leitura do coletor."
60      ::= { tccEntradaParaProcessos 1 }
61
62  pID OBJECT-TYPE
63      SYNTAX      Integer32 (0..2147483647)
64      MAX-ACCESS  read-only
65      STATUS      current
66      DESCRIPTION
67       "Armazena o ID do processo que está radando na máquina local."
68      ::= { tccEntradaParaProcessos 2 }
69
70  pDataHoraDaColeta OBJECT-TYPE
71      SYNTAX      OCTET STRING
72      MAX-ACCESS  read-only
73      STATUS      current
74      DESCRIPTION
75       "Armazena a data e hora da coleta. Esta data está no padrão
76           yyyymmdd hh:ii:ss"
77      ::= { tccEntradaParaProcessos 3 }
78
79  pNome OBJECT-TYPE
80      SYNTAX      OCTET STRING
81      MAX-ACCESS  read-only
82      STATUS      current
83      DESCRIPTION
84       "Armazena o nome do processo que está radando na máquina local."
85      ::= { tccEntradaParaProcessos 4 }
86
87  pUsuario OBJECT-TYPE
88      SYNTAX      OCTET STRING
89      MAX-ACCESS  read-only
90      STATUS      current
91      DESCRIPTION
92       "Armazena o usuário do processo que está radando na máquina local."
93      ::= { tccEntradaParaProcessos 5 }
94
95  pUtilizacaoDaCPU OBJECT-TYPE
96      SYNTAX      OCTET STRING
97      MAX-ACCESS  read-only
98      STATUS      current
99      DESCRIPTION
100      "Armazena o percentual de utilização da CPU pelo processo
101          que está radando na máquina local."
102     ::= { tccEntradaParaProcessos 6 }
103
104 pUtilizacaoDaMemoria OBJECT-TYPE
105     SYNTAX      OCTET STRING
106     MAX-ACCESS  read-only
107     STATUS      current
108     DESCRIPTION
109      "Armazena o percentual de utilização da memória pelo processo
110          que está radando na máquina local."
111     ::= { tccEntradaParaProcessos 7 }
112
113
114
115 tccTemperatureTable OBJECT-TYPE
```

```
116     SYNTAX      SEQUENCE OF TccEntradaParaTemperatura
117     MAX-ACCESS  not-accessible
118     STATUS      current
119     DESCRIPTION
120      "Armazena temporariamente informações sobre a temperatura da CPU."
121     ::= { tccTabela 2 }
122
123 tccEntradaParaTemperatura OBJECT-TYPE
124     SYNTAX      TccEntradaParaTemperatura
125     MAX-ACCESS  not-accessible
126     STATUS      current
127     DESCRIPTION
128      "Identifica uma linha da tabela"
129     INDEX   { pIDDaLinha }
130     ::= {tccTemperatureTable 1 }
131
132 TccEntradaParaTemperatura ::= SEQUENCE {
133      tIDDaLinha          Integer32,
134      tDataHoraDaColeta OCTET STRING,
135      tTemperatura            Integer32
136 }
137
138 tIDDaLinha OBJECT-TYPE
139     SYNTAX      Integer32 (0..2147483647)
140     MAX-ACCESS  read-only
141     STATUS      current
142     DESCRIPTION
143      "Número da linha. Este valor é reiniciado pelo agente
144          conforme as requisiçoes de leitura do coletor."
145     ::= { tccEntradaParaTemperatura 1 }
146
147 tDataHoraDaColeta OBJECT-TYPE
148     SYNTAX      OCTET STRING
149     MAX-ACCESS  read-only
150     STATUS      current
151     DESCRIPTION
152      "Armazena a data e hora da coleta. Esta data está no padrão
153          yyyymmdd hh:ii:ss"
154     ::= { tccEntradaParaTemperatura 2 }
155
156 tTemperatura OBJECT-TYPE
157     SYNTAX      Integer32
158     MAX-ACCESS  read-only
159     STATUS      current
160     DESCRIPTION
161      "Armazena a temperatura da CPU."
162     ::= { tccEntradaParaTemperatura 3 }
163
164 tccPartitionTable OBJECT-TYPE
165     SYNTAX      SEQUENCE OF TccEntradaParaParticoes
166     MAX-ACCESS  not-accessible
167     STATUS      current
168     DESCRIPTION
169      "Armazena temporariamente informações sobre as partições do disco."
170     ::= { tccTabela 3 }
171
172 tccEntradaParaParticoes OBJECT-TYPE
173     SYNTAX      TccEntradaParaParticoes
174     MAX-ACCESS  not-accessible
175     STATUS      current
176     DESCRIPTION
177      "Identifica uma linha da tabela"
178     INDEX   { aIDDaLinha }
179     ::= {tccPartitionTable 1 }
180
181 TccEntradaParaParticoes ::= SEQUENCE {
182      aIDDaLinha          Integer32,
```

```
183      aDataHoraDaColeta OCTET STRING,
184      aNome       OCTET STRING,
185      aPontoDeMontagem        OCTET STRING,
186      aTotalEmKb  OCTET STRING,
187      aUsadoEmKb  OCTET STRING
188 }
189
190 aIDDaLinha OBJECT-TYPE
191     SYNTAX      Integer32 (0..2147483647)
192     MAX-ACCESS  read-only
193     STATUS      current
194     DESCRIPTION
195      "Número da linha. Este valor é reiniciado pelo agente
196          conforme as requisiçoes de leitura do coletor."
197     ::= { tccEntradaParaParticoes 1 }
198
199 aDataHoraDaColeta OBJECT-TYPE
200     SYNTAX      OCTET STRING
201     MAX-ACCESS  read-only
202     STATUS      current
203     DESCRIPTION
204      "Armazena a data e hora da coleta. Esta data está no padrão
205          yyyymmdd hh:ii:ss"
206     ::= { tccEntradaParaParticoes 2 }
207
208 aNome OBJECT-TYPE
209     SYNTAX      OCTET STRING
210     MAX-ACCESS  read-only
211     STATUS      current
212     DESCRIPTION
213      "Armazena o nome da partição."
214     ::= { tccEntradaParaParticoes 3 }
215
216 aPontoDeMontagem OBJECT-TYPE
217     SYNTAX      OCTET STRING
218     MAX-ACCESS  read-only
219     STATUS      current
220     DESCRIPTION
221      "Armazena o ponto de montagem da partição."
222     ::= { tccEntradaParaParticoes 4 }
223
224 aTotalEmKb OBJECT-TYPE
225     SYNTAX      OCTET STRING
226     MAX-ACCESS  read-only
227     STATUS      current
228     DESCRIPTION
229      "Armazena o total em Kb de armazenamento da partição"
230     ::= { tccEntradaParaParticoes 5 }
231
232 aUsadoEmKb OBJECT-TYPE
233     SYNTAX      OCTET STRING
234     MAX-ACCESS  read-only
235     STATUS      current
236     DESCRIPTION
237      "Armazena o total usado em Kb da partição"
238     ::= { tccEntradaParaParticoes 6 }
239
240 END
```

APÊNDICE B Código fonte do agente

Neste apêndice pode-se visualizar cada arquivo fonte da aplicação agented. O agente é composto por 22 arquivos fontes, onde 20 são mostrados abaixo e 2 (*util.h* e *util.c*) são descritos no apêndice 4.

- *agented.c*: fonte principal que roda o agente extensível para registrar as tabelas descritas na MIB e permitir a comunicação com o agente. Também executa um fluxo de código em paralelo para armazenar as informações extraías na MIB.

```
1   #include "agented.h"
2
3
4   RETSIGTYPE
5   stop_server(int a) {
6       keep_running = 0;
7   }
8
9   int main (int argc, char **argv)
10  {
11      agtConfiguracao *agtConf;
12      agtConf = (agtConfiguracao*) malloc(sizeof(agtConfiguracao));
13
14      //Busca as configurações do coletor
15      cfg_opt_t opts[] = {
16                  CFG_SIMPLE_INT ("agtTempoDeExecucaoDoLaco", &agtConf-
    >agtTempoDeExecucaoDoLaco),
17                  CFG_SIMPLE_INT ("agtTempoDeCapturaParaProcesso", &agtConf-
    >agtTempoDeCapturaProcesso),
18                  CFG_SIMPLE_INT ("agtTempoDeCapturaParaTemperatura", &agtConf-
    >agtTempoDeCapturaTemperatura),
19                  CFG_SIMPLE_INT ("agtTempoDeCapturaParaParticao", &agtConf-
    >agtTempoDeCapturaParticao),
20                  CFG_SIMPLE_STR ("agtProcTemperatura", &agtConf-
    >agtProcTemperatura),
21                  CFG_SIMPLE_INT ("agtExibeNoticia", &agtConf->agtExibeNoticia),
22                  CFG_END()
23                  };
24      utlLeConfiguracao(AGT_ARQUIVO_CONFIGURACAO, opts);
25
26      UTL_EXIBE_NOTICIA = agtConf->agtExibeNoticia;
27
```

```
28
29
30      int background = 0;
31      int syslog = 0;
32
33      snmp_enable_stderrlog();
34
35
36      netsnmp_ds_set_boolean(NETSNMP_DS_APPLICATION_ID, NETSNMP_DS_AGENT_ROLE, 1);
37
38      if (background && netsnmp_daemonize(1, !syslog))
39          exit(1);
40
41      SOCK_STARTUP;
42
43      DEBUGMSG(("Before agent library init","\n"));
44      init_agent("agented");
45
46
47      init_tccProcessTable();
48      init_tccTemperatureTable();
49      init_tccPartitionTable();
50
51      init_snmp("agented");
52
53      keep_running = 1;
54      signal(SIGTERM, stop_server);
55      signal(SIGINT, stop_server);
56
57      snmp_log(LOG_INFO,"agented is up and running.\n");
58
59      pthread_t captura;
60      pthread_create(&captura,NULL,(void *(*)(void *))&agtIniciaCaptura,(void *)
    agtConf);
61
62      while(keep_running)
63      {
64          agent_check_and_process(1);
65      }
66
67      snmp_shutdown("agented");
68      SOCK_CLEANUP;
69
70      free(agtConf);
71
72      return 0;
73  }
74
75  void agtIniciaCaptura(void *conf)
76  {
77
78      int tempoEmSegundos;
79      utlTempoEmSegundos(&tempoEmSegundos);
80      int executarProcesso     = tempoEmSegundos;
81      int executarTemperatura = tempoEmSegundos;
82      int executarParticao     = tempoEmSegundos;
83
84      utlDebug("Iniciando monitoração pelo agente.", UTL_NOTICIA);
85      struct agtProcesso *ok;
86
87      agtConfiguracao *agtConf;
88      agtConf = conf;
89
90      utlObtemTempoDecorrido();
91      agtAtribuiListaDeProcessos();
92      int x=1;
93
```

```
94      while (1)
95      {
96          //Tempo de espera para execução do laço
97          sleep(agtConf->agtTempoDeExecucaoDoLaco);
98
99          char dataHora[40];
100         utlDataHoraAtual(dataHora);
101
102         if (utlExecutarColeta(executarProcesso))
103         {
104             //Atribui o tempo para a próxima leitura
105             executarProcesso += agtConf->agtTempoDeCapturaProcesso;
106
107             float tempoDeEspera = utlObtemTempoDecorrido();
108             float percentualTotalCPU = 0;
109             agtAtribuiListaDeProcessos();
110
111
112             //Extrai processos
113             agtPonteiroParaInicio();
114             do
115             {
116                 ok = agtPegaProcessoAtual();
117                 float percentualCPU     = (ok->jiffies / (HZ * tempoDeEspera)) *
    100;
118                 float percentualMemoria = (float) (ok->resident / (float)
    utlTotalMemKB()) * 100;
119                 percentualTotalCPU += percentualCPU;
120                 if (ok->jiffies > 0)
121                 {
122                     char buf1[30];
123                     char buf2[30];
124
125                     sprintf(buf1, "%f", percentualCPU);
126                     sprintf(buf2, "%f", percentualMemoria);
127                     agtAdicionaProcessoNaMIB(ok->pid, dataHora, ok->comm, ok-
    >usuario, buf1, buf2);
128                     x++;
129                 }
130             } while (agtProximoProcesso());
131         }
132
133         //Extrai temperatura
134         if (agtConf->agtProcTemperatura != NULL)
135         {
136             if (utlExecutarColeta(executarTemperatura))
137             {
138                 //Atribui o tempo para a próxima leitura
139                 executarTemperatura += agtConf->agtTempoDeCapturaTemperatura;
140
141                 int temperatura;
142                 utlLeTemperatura(&temperatura, agtConf->agtProcTemperatura);
143                 agtAdicionaTemperaturaNaMIB(dataHora, temperatura);
144             }
145         }
146
147         //Extrai dados das particões do disco
148         if (utlExecutarColeta(executarParticao))
149         {
150             //Atribui o tempo para a próxima leitura
151             executarParticao += agtConf->agtTempoDeCapturaParticao;
152
153             agtListaDeParticoes *agtLParticoes, *agtLParticoesAux;
154             agtLParticoes = (agtListaDeParticoes*)
    malloc(sizeof(agtListaDeParticoes));
155
156             agtLeParticoes(agtLParticoes);
```

```
157             for (agtLParticoesAux = agtLParticoes; agtLParticoesAux !=
    (agtListaDeParticoes*) NULL;  agtLParticoesAux = agtLParticoesAux->proximo)
158             {
159                 char buf1[30];
160                 char buf2[30];
161                 sprintf(buf1, "%f", agtLParticoesAux->totalEmKb);
162                 sprintf(buf2, "%f", agtLParticoesAux->usadoEmKb);
163 
164                 agtAdicionaParticoesNaMIB(dataHora, agtLParticoesAux->nome,
    agtLParticoesAux->pontoDeMontagem, buf1, buf2);
165             }
166             agtRemoverListaDeParticoes(agtLParticoes);
167         }
168     }
169 }
```

- *agented.h*: define funções e estruturas encontradas no arquivo *agented.c*.

```
1   #ifndef AGENTED_H
2   #define AGENTED_H
3   
4   #include <net-snmp/net-snmp-config.h>
5   #include <net-snmp/net-snmp-includes.h>
6   #include <net-snmp/agent/net-snmp-agent-includes.h>
7   
8   #include <signal.h>
9   #include <pthread.h>
10  
11  #include <stdio.h>
12  #include <stdlib.h>
13  #include <unistd.h>
14  #include <dirent.h>
15  
16  #include <string.h>
17  #include <sys/time.h>
18  #include <time.h>
19  #include <sys/param.h>
20  #include <pwd.h>
21  #include <sys/stat.h>
22  
23  static int keep_running;
24  
25  #define UTLCMPAGENTE 1
26  #include "util.h"
27  
28  //Define o nome do arquivo de configuração do coletor.
29  #define AGT_ARQUIVO_CONFIGURACAO "agented.conf"
30  #define PAGE_SIZE ( sysconf(_SC_PAGESIZE) / 1024 )
31  
32  //Estrutura que armazena os dados de configuração do coletor
33  typedef struct agtConfiguracao_
34  {
35      int agtTempoDeExecucaoDoLaco;
36      int agtTempoDeCapturaProcesso;
37      int agtTempoDeCapturaTemperatura;
38      int agtTempoDeCapturaParticao;
39      char *agtProcTemperatura;
40      int agtExibeNoticia;
41  
```

```
42 } agtConfiguracao;
43
44 void agtIniciaCaptura();
45
46 #include "tccProcessTable.h"
47 #include "tccTemperatureTable.h"
48 #include "tccPartitionTable.h"
49 #include "agtListaDeProcessos.h"
50 #include "agtParticoes.h"
51
52 #endif
```

- *tccProcessTable.h*: define as funções encontradas no arquivo *tccProcessTable.c*. Este arquivo é gerado com o auxílio da aplicação mib2c. Esta aplicação lê uma MIB e gera o código fonte em C para inicializar e registrar um objeto.

```
1  /*
2   * Note: this file originally auto-generated by mib2c using
3   *  : mib2c.create-dataset.conf,v 5.4 2004/02/02 19:06:53 rstory Exp $
4   */
5  #ifndef TCCPROCESSTABLE_H
6  #define TCCPROCESSTABLE_H
7
8  /* function declarations */
9  void init_tccProcessTable(void);
10 void initialize_table_tccProcessTable(void);
11 Netsnmp_Node_Handler tccProcessTable_handler;
12
13 /* column number definitions for table tccProcessTable */
14        #define COLUMN_PIDDALINHA               1
15        #define COLUMN_PID              2
16        #define COLUMN_PDATAHORADACOLETA        3
17        #define COLUMN_PNOME            4
18        #define COLUMN_PUSUARIO         5
19        #define COLUMN_PUTILIZACAODACPU         6
20        #define COLUMN_PUTILIZACAODAMEMORIA             7
21 #endif /* TCCPROCESSTABLE_H */
```

- *tccProcessTable.c*: utilizado para inicializar e registrar o objeto *tccProcessTable*. Este arquivo é gerado com o auxílio da aplicação mib2c.

```
1  /*
2   * Note: this file originally auto-generated by mib2c using
```

```
3    *         : mib2c.create-dataset.conf,v 5.4 2004/02/02 19:06:53 rstory Exp $
4    */
5
6   #include <net-snmp/net-snmp-config.h>
7   #include <net-snmp/net-snmp-includes.h>
8   #include <net-snmp/agent/net-snmp-agent-includes.h>
9   #include "tccProcessTable.h"
10
11  netsnmp_table_data_set *table_set;
12  #include "agtTabelaDeProcessos_.h"
13
14  /** Initialize the tccProcessTable table by defining its contents and how it's
    structured */
15  void
16  initialize_table_tccProcessTable(void)
17  {
18      static oid tccProcessTable_oid[] = {1,3,6,1,4,1,8072,1000,1,1};
19      size_t tccProcessTable_oid_len = OID_LENGTH(tccProcessTable_oid);
20
21      /* create the table structure itself */
22      table_set = netsnmp_create_table_data_set("tccProcessTable");
23
24      /* comment this out or delete if you don't support creation of new rows */
25      //table_set->allow_creation = 1;
26
27      /**************************************************
28       * Adding indexes
29       */
30      DEBUGMSGTL(("initialize_table_tccProcessTable",
31                  "adding indexes to table tccProcessTable\n"));
32      netsnmp_table_set_add_indexes(table_set,
33                            ASN_INTEGER,  /* index: pIDDaLinha */
34                            0);
35
36      DEBUGMSGTL(("initialize_table_tccProcessTable",
37                  "adding column types to table tccProcessTable\n"));
38      netsnmp_table_set_multi_add_default_row(table_set,
39                                              COLUMN_PIDDALINHA, ASN_INTEGER, 1,
40                                              NULL, 0,
41                                              COLUMN_PID, ASN_INTEGER, 1,
42                                              NULL, 0,
43                                              COLUMN_PDATAHORADACOLETA,
    ASN_OCTET_STR, 1,
44                                              NULL, 0,
45                                              COLUMN_PNOME, ASN_OCTET_STR, 1,
46                                              NULL, 0,
47                                              COLUMN_PUSUARIO, ASN_OCTET_STR, 1,
48                                              NULL, 0,
49                                              COLUMN_PUTILIZACAODACPU,
    ASN_OCTET_STR, 1,
50                                              NULL, 0,
51                                              COLUMN_PUTILIZACAODAMEMORIA,
    ASN_OCTET_STR, 1,
52                                              NULL, 0,
53                                  0);
54
55      /* registering the table with the master agent */
56      /* note: if you don't need a subhandler to deal with any aspects
57         of the request, change tccProcessTable_handler to "NULL" */
58      netsnmp_register_table_data_set(netsnmp_create_handler_registration("tccProce
    ssTable", tccProcessTable_handler,
59                                                          tccProcessTable_oid,
60                                                          tccProcessTable_oid_len,
61                                                          HANDLER_CAN_RWRITE),
62                                  table_set, NULL);
63  }
64
```

```
65 /** Initializes the tccProcessTable module */
66 void
67 init_tccProcessTable(void)
68 {
69
70   /* here we initialize all the tables we're planning on supporting */
71     initialize_table_tccProcessTable();
72 }
73
74 /** handles requests for the tccProcessTable table, if anything else needs to be
   done */
75 int
76 tccProcessTable_handler(
77     netsnmp_mib_handler               *handler,
78     netsnmp_handler_registration      *reginfo,
79     netsnmp_agent_request_info        *reqinfo,
80     netsnmp_request_info              *requests) {
81     /* perform anything here that you need to do.  The requests have
82        already been processed by the master table_dataset handler, but
83        this gives you chance to act on the request in some other way
84        if need be. */
85     return agtTabelaDeProcessosHandler(handler, reginfo, reqinfo, requests);
86 }
```

- *tccTemperatureTable.h*: define as funções encontradas no arquivo *tccTemperatureTable.c*. Este arquivo é gerado com o auxílio da aplicação mib2c.

```
1  /*
2   * Note: this file originally auto-generated by mib2c using
3   *  : mib2c.create-dataset.conf,v 5.4 2004/02/02 19:06:53 rstory Exp $
4   */
5  #ifndef TCCTEMPERATURETABLE_H
6  #define TCCTEMPERATURETABLE_H
7
8  /* function declarations */
9  void init_tccTemperatureTable(void);
10 void initialize_table_tccTemperatureTable(void);
11 Netsnmp_Node_Handler tccTemperatureTable_handler;
12
13 /* column number definitions for table tccTemperatureTable */
14        #define COLUMN_TIDDALINHA               1
15        #define COLUMN_TDATAHORADACOLETA        2
16        #define COLUMN_TTEMPERATURA             3
17 #endif /* TCCTEMPERATURETABLE_H */
```

- tccTemperatureTable.c: utilizado para inicializar e registrar o objeto *tccTemperatureTable*. Este arquivo é gerado com o auxílio da aplicação mib2c.

```
1   /*
2    * Note: this file originally auto-generated by mib2c using
3    *        : mib2c.create-dataset.conf,v 5.4 2004/02/02 19:06:53 rstory Exp $
4    */
5
6   #include <net-snmp/net-snmp-config.h>
7   #include <net-snmp/net-snmp-includes.h>
8   #include <net-snmp/agent/net-snmp-agent-includes.h>
9   #include "tccTemperatureTable.h"
10
11  netsnmp_table_data_set *t_table_set;
12  #include "agtTabelaDeTemperatura_.h"
13
14  /** Initialize the tccTemperatureTable table by defining its contents and how
    it's structured */
15  void
16  initialize_table_tccTemperatureTable(void)
17  {
18      static oid tccTemperatureTable_oid[] = {1,3,6,1,4,1,8072,1000,1,2};
19      size_t tccTemperatureTable_oid_len = OID_LENGTH(tccTemperatureTable_oid);
20
21      /* create the table structure itself */
22      t_table_set = netsnmp_create_table_data_set("tccTemperatureTable");
23
24      /* comment this out or delete if you don't support creation of new rows */
25      //t_table_set->allow_creation = 1;
26
27      /**************************************************
28       * Adding indexes
29       */
30      DEBUGMSGTL(("initialize_table_tccTemperatureTable",
31                  "adding indexes to table tccTemperatureTable\n"));
32      netsnmp_table_set_add_indexes(t_table_set,
33                            ASN_INTEGER,  /* index: pIDDaLinha */
34                            0);
35
36      DEBUGMSGTL(("initialize_table_tccTemperatureTable",
37                  "adding column types to table tccTemperatureTable\n"));
38      netsnmp_table_set_multi_add_default_row(t_table_set,
39                                              COLUMN_TIDDALINHA, ASN_INTEGER, 1,
40                                              NULL, 0,
41                                              COLUMN_TDATAHORADACOLETA,
    ASN_OCTET_STR, 1,
42                                              NULL, 0,
43                                              COLUMN_TTEMPERATURA, ASN_INTEGER, 1,
44                                              NULL, 0,
45                                  0);
46
47      /* registering the table with the master agent */
48      /* note: if you don't need a subhandler to deal with any aspects
49         of the request, change tccTemperatureTable_handler to "NULL" */
50      netsnmp_register_table_data_set(netsnmp_create_handler_registration("tccTempe
    ratureTable", tccTemperatureTable_handler,
51                                                              tccTemperatureTable_oid,
52                                                              tccTemperatureTable_oid_l
    en,
53                                                              HANDLER_CAN_RWRITE),
```

```
54                              t_table_set, NULL);
55  }
56
57  /** Initializes the tccTemperatureTable module */
58  void
59  init_tccTemperatureTable(void)
60  {
61
62    /* here we initialize all the tables we're planning on supporting */
63      initialize_table_tccTemperatureTable();
64  }
65
66  /** handles requests for the tccTemperatureTable table, if anything else needs to
    be done */
67  int
68  tccTemperatureTable_handler(
69      netsnmp_mib_handler               *handler,
70      netsnmp_handler_registration      *reginfo,
71      netsnmp_agent_request_info        *reqinfo,
72      netsnmp_request_info              *requests) {
73      /* perform anything here that you need to do.  The requests have
74         already been processed by the master table_dataset handler, but
75         this gives you chance to act on the request in some other way
76         if need be. */
77      return agtTabelaDeTemperaturaHandler(handler, reginfo, reqinfo, requests);
78  }
```

- *tccPartitionTable.h*: define as funções encontradas no arquivo *tccPartitionTable.c*. Este arquivo é gerado com o auxílio da aplicação mib2c.

```
1   /*
2    * Note: this file originally auto-generated by mib2c using
3    *  : mib2c.create-dataset.conf,v 5.4 2004/02/02 19:06:53 rstory Exp $
4    */
5   #ifndef TCCPARTITIONTABLE_H
6   #define TCCPARTITIONTABLE_H
7
8   /* function declarations */
9   void init_tccPartitionTable(void);
10  void initialize_table_tccPartitionTable(void);
11  Netsnmp_Node_Handler tccPartitionTable_handler;
12
13  /* column number definitions for table tccPartitionTable */
14          #define COLUMN_AIDDALINHA               1
15          #define COLUMN_ADATAHORADACOLETA        2
16          #define COLUMN_ANOME            3
17          #define COLUMN_APONTODEMONTAGEM         4
18          #define COLUMN_ATOTALEMKB               5
19          #define COLUMN_AUSADOEMKB               6
20  #endif /* TCCPARTITIONTABLE_H */
```

- *tccPartitionTable.c*: utilizado para inicializar e registrar o objeto *tccPartitionTable*. Este arquivo é gerado com o auxílio da aplicação mib2c.

```
1   /*
2    * Note: this file originally auto-generated by mib2c using
3    *        : mib2c.create-dataset.conf,v 5.4 2004/02/02 19:06:53 rstory Exp $
4    */
5
6   #include <net-snmp/net-snmp-config.h>
7   #include <net-snmp/net-snmp-includes.h>
8   #include <net-snmp/agent/net-snmp-agent-includes.h>
9   #include "tccPartitionTable.h"
10
11  netsnmp_table_data_set *a_table_set;
12  #include "agtTabelaDeParticoes_.h"
13
14  /** Initialize the tccPartitionTable table by defining its contents and how it's
    structured */
15  void
16  initialize_table_tccPartitionTable(void)
17  {
18      static oid tccPartitionTable_oid[] = {1,3,6,1,4,1,8072,1000,1,3};
19      size_t tccPartitionTable_oid_len = OID_LENGTH(tccPartitionTable_oid);
20
21      /* create the table structure itself */
22      a_table_set = netsnmp_create_table_data_set("tccPartitionTable");
23
24      /* comment this out or delete if you don't support creation of new rows */
25      //a_table_set->allow_creation = 1;
26
27      /**************************************************
28       * Adding indexes
29       */
30      DEBUGMSGTL(("initialize_table_tccPartitionTable",
31                  "adding indexes to table tccPartitionTable\n"));
32      netsnmp_table_set_add_indexes(a_table_set,
33                            ASN_INTEGER,  /* index: aIDDaLinha */
34                            0);
35
36      DEBUGMSGTL(("initialize_table_tccPartitionTable",
37                  "adding column types to table tccPartitionTable\n"));
38      netsnmp_table_set_multi_add_default_row(a_table_set,
39                                              COLUMN_AIDDALINHA, ASN_INTEGER, 1,
40                                              NULL, 0,
41                                              COLUMN_ADATAHORADACOLETA,
    ASN_OCTET_STR, 1,
42                                              NULL, 0,
43                                              COLUMN_ANOME, ASN_OCTET_STR, 1,
44                                              NULL, 0,
45                                              COLUMN_APONTODEMONTAGEM,
    ASN_OCTET_STR, 1,
46                                              NULL, 0,
47                                              COLUMN_ATOTALEMKB, ASN_OCTET_STR, 1,
48                                              NULL, 0,
49                                              COLUMN_AUSADOEMKB, ASN_OCTET_STR, 1,
50                                              NULL, 0,
51                                  0);
52
53      /* registering the table with the master agent */
54      /* note: if you don't need a subhandler to deal with any aspects
55         of the request, change tccPartitionTable_handler to "NULL" */
56      netsnmp_register_table_data_set(netsnmp_create_handler_registration("tccParti
```

```
   tionTable", tccPartitionTable_handler,
57                                                      tccPartitionTable_oid,
58                                                      tccPartitionTable_oid_len
   ,
59                                                      HANDLER_CAN_RWRITE),
60                             a_table_set, NULL);
61 }
62
63 /** Initializes the tccPartitionTable module */
64 void
65 init_tccPartitionTable(void)
66 {
67
68   /* here we initialize all the tables we're planning on supporting */
69     initialize_table_tccPartitionTable();
70 }
71
72 /** handles requests for the tccPartitionTable table, if anything else needs to
   be done */
73 int
74 tccPartitionTable_handler(
75     netsnmp_mib_handler               *handler,
76     netsnmp_handler_registration      *reginfo,
77     netsnmp_agent_request_info        *reqinfo,
78     netsnmp_request_info              *requests) {
79     /* perform anything here that you need to do.  The requests have
80        already been processed by the master table_dataset handler, but
81        this gives you chance to act on the request in some other way
82        if need be. */
83     return agtTabelaDeParticoesHandler(handler, reginfo, reqinfo, requests);
84 }
```

- *agtListaDeProcessos.h*: define as funções encontradas no arquivo *tccListaDeProcessos.c*.

```
1  #ifndef AGTLISTADEPROCESSO_H
2  #define AGTLISTADEPROCESSO_H
3
4  #include "util.h"
5  #include "agtProcesso.h"
6
7  int agtAtribuiListaDeProcessos();
8
9  #include "agtListaDeProcessos.c"
10
11 #endif
```

- *agtListaDeProcessos.c*: atribui e atualiza as informações dos processos.

```
1   int agtAtribuiListaDeProcessos()
2   {
3       struct dirent* entrada;
4       struct agtProcesso *teste;
5       int term;
6       DIR* dir;
7       int pid;
8
9       dir = opendir(PROC_DIR);
10      if (!dir)
11      {
12          perror(PROC_DIR);
13          return 0;
14      }
15
16      agtPonteiroParaInicio();
17      while ((entrada = readdir(dir)) != NULL)
18      {
19          char *nome = entrada->d_name;
20          pid = atoi(nome);
21          if (pid > 0 )
22          {
23              term=1;
24              while (term == 1)
25              {
26                  teste = agtPegaProcessoAtual();
27                  term = 0;
28
29                  //Não existe mais processo na lista. Assim, adiciona.
30                  if (qtde_de_processos == 0)
31                  {
32                      agtAdicionaProcesso(pid);
33                  }
34                  else if (!agtExisteProximoProcesso() && teste->pid < pid)
35                  {
36                      agtAdicionaProcesso(pid);
37                  }
38                  else
39                  {
40                      if (teste->pid == pid)
41                      {
42                          agtAtualizaProcesso(pid);
43                          agtProximoProcesso();
44                      }
45                      else //Os processos não batem
46                      {
47                          //novo processos
48                          if (teste->pid > pid)
49                          {
50                              adicionaAntes(pid);
51                              agtProximoProcesso();
52                          }
53                          else if (teste->pid < pid)//processo morreu
54                          {
55                              term = 1;
56                              agtRemoveProcessoAtual();
57                          }
58                      }
59                  }
60              }
61          }
62      }
63
64      //Remove processo do final da lista que não estão rodando no sistema
65      do
66      {
67          teste = agtPegaProcessoAtual();
```

```
68          if (teste->pid > pid)
69          {
70              agtRemoveProcessoAtual();
71          }
72      } while (agtProximoProcesso());
73
74      closedir(dir);
75
76      return 1;
77  }
```

- *agtProcesso.h*: define funções e estruturas encontradas no arquivo *agtProcesso.c*.

```
1   #ifndef AGTPROCESSO_H
2   #define AGTPROCESSO_H
3
4   #include "util.h"
5
6   struct agtProcesso {
7       /* Atributos da CPU */
8       int pid;
9       char comm[64];
10      unsigned long utime;
11      unsigned long stime;
12      float jiffies;
13      float ultimosJiffies;
14      char usuario[64];
15
16      /* Atributos memória */
17      int size;
18      int resident;
19      int share;
20      int text;
21      int lib;
22      int data;
23      int dt;
24
25      struct agtProcesso *processo_anterior;
26      struct agtProcesso *prox_processo;
27  };
28
29  struct agtProcesso *processo_atual = (struct agtProcesso*) NULL,
    *processo_inicial = (struct agtProcesso*) NULL;
30  int qtde_de_processos = 0;
31
32
33  void agtPonteiroParaInicio();
34  struct agtProcesso *agtPegaProcessoAtual();
35  int agtExisteProximoProcesso();
36  int agtProximoProcesso();
37  int agtLeProc(int pid);
38  int agtAdicionaProcesso(int pid);
39  int agtAtualizaProcesso(int pid);
40  int agtRemoveProcessoAtual();
41
42  #include "agtProcesso.c"
```

```
43
44
45  #endif
```

- *agtProcesso.c*: possui funções para manipular a lista de processos e para extrair informações de memória e processamento.

```
1   /* Funções para controle da estrutura */
2   void agtPonteiroParaInicio()
3   {
4       processo_atual = processo_inicial;
5   }
6
7   struct agtProcesso *agtPegaProcessoAtual()
8   {
9       return processo_atual;
10  }
11
12  int agtExisteProximoProcesso()
13  {
14      if (processo_atual->prox_processo != (struct agtProcesso*) NULL)
15      {
16          return 1;
17      }
18      return 0;
19  }
20
21  int agtProximoProcesso()
22  {
23      if (agtExisteProximoProcesso())
24      {
25          processo_atual = processo_atual->prox_processo;
26          return 1;
27      }
28      return 0;
29  }
30
31  int agtLeProc(int pid)
32  {
33      FILE* status;
34      char caminho[UTL_TAM_MAX_P + 1];
35
36      long int luNull;
37      int dNull;
38      char cNull;
39
40      snprintf(caminho, UTL_TAM_MAX_P, "%s/%d/stat", PROC_DIR, pid);
41      status = fopen(caminho, "r");
42      if (status)
43      {
44
45          fscanf(status, "%d %s %c %d "
46                         "%d %d %d %d "
47                         "%lu %lu %lu %lu "
48                         "%lu %lu %lu",
49                          &processo_atual->pid, &processo_atual->comm, &cNull,
    &dNull,
```

```
50                          &dNull, &dNull, &dNull, &dNull,
51                          &luNull, &luNull, &luNull, &luNull,
52                          &luNull, &processo_atual->utime, &processo_atual->stime);
53
54
55          utlUsuarioDoProcesso(processo_atual->pid, processo_atual->usuario);
56
57          float tempoTotal = (float) processo_atual->utime + processo_atual->stime;
58          processo_atual->jiffies = tempoTotal - processo_atual->ultimosJiffies;
59          processo_atual->ultimosJiffies = tempoTotal;
60      }
61      fclose(status);
62
63      //Memória
64      snprintf(caminho, UTL_TAM_MAX_P, "%s/%d/statm", PROC_DIR, pid);
65      status = fopen(caminho, "r");
66      if (status)
67      {
68          fscanf(status, "%d %d %d %d %d %d %d",
69                  &processo_atual->size,
70                  &processo_atual->resident,
71                  &processo_atual->share,
72                  &processo_atual->text,
73                  &processo_atual->lib,
74                  &processo_atual->data,
75                  &processo_atual->dt);
76
77          processo_atual->resident = processo_atual->resident * PAGE_SIZE;
78      }
79      fclose(status);
80
81      return 1;
82  }
83
84  int agtAdicionaProcesso(int pid)
85  {
86      utlDebug("Adicionando novo processo.", UTL_NOTICIA);
87
88      struct agtProcesso *novo_processo;
89      novo_processo = (struct agtProcesso*) malloc(sizeof(struct agtProcesso));
90
91      /* Verificar se é o primeiro processo para manter o controle de início da
    lista */
92      if (processo_inicial == (struct agtProcesso*) NULL)
93      {
94          utlDebug("Primeiro processo.", UTL_NOTICIA);
95          novo_processo->prox_processo = (struct agtProcesso*) NULL;
96          processo_inicial = processo_atual = novo_processo;
97      }
98      else
99      {
100         /* Coloca o ponteiro atual com sendo o último da lista */
101         while (agtExisteProximoProcesso())
102         {
103             processo_atual = processo_atual->prox_processo;
104         }
105
106         novo_processo->processo_anterior = processo_atual;
107         processo_atual->prox_processo = novo_processo;
108
109         //Atribui ao processo atual o novo processo
110         processo_atual = novo_processo;
111         processo_atual->prox_processo = (struct agtProcesso*) NULL;
112     }
113
114     qtde_de_processos++;
115     agtLeProc(pid);
```

```
116
117     return 1;
118 }
119
120 int adicionaAntes(pid)
121 {
122     utlDebug("Adicionando novo processo antes do atual.", UTL_NOTICIA);
123
124     struct agtProcesso *novo_processo;
125     novo_processo = (struct agtProcesso*) malloc(sizeof(struct agtProcesso));
126
127     novo_processo->processo_anterior = processo_atual->processo_anterior;
128     novo_processo->prox_processo = processo_atual;
129
130     processo_atual->processo_anterior->prox_processo = novo_processo;
131
132     processo_atual->processo_anterior = novo_processo;
133
134
135     processo_atual = novo_processo;
136
137     qtde_de_processos++;
138     agtLeProc(pid);
139
140     return 1;
141 }
142
143 int agtAtualizaProcesso(int pid)
144 {
145     agtLeProc(pid);
146
147     return 1;
148 }
149
150 int agtRemoveProcessoAtual()
151 {
152     utlDebug("Removendo processo.", UTL_NOTICIA);
153
154     struct agtProcesso *processo_auxiliar = (struct agtProcesso*) NULL;
155
156     if (processo_atual->processo_anterior != (struct agtProcesso*) NULL)
157     {
158         processo_atual->processo_anterior->prox_processo = processo_atual-
    >prox_processo;
159         processo_auxiliar = processo_atual->processo_anterior;
160     }
161
162     if (processo_atual->prox_processo != (struct agtProcesso*) NULL)
163     {
164         processo_atual->prox_processo->processo_anterior = processo_atual-
    >processo_anterior;
165         processo_auxiliar = processo_atual->prox_processo;
166     }
167     free(processo_atual);
168     processo_atual = processo_auxiliar;
169     processo_auxiliar = (struct agtProcesso*) NULL;
170     free(processo_auxiliar);
171     qtde_de_processos--;
172
173     return 1;
174 }
```

- *agtParticoes.h*: define funções e estruturas encontradas no arquivo *agtParticoes.c*.

```
1   #ifndef AGTPARTICOES_H
2   #define AGTPARTICOES_H
3
4   #include <mntent.h>
5   #include <sys/statfs.h>
6
7   typedef struct agtListaDeParticoes_
8   {
9       char nome[UTL_TAM_MAX_P];
10      char pontoDeMontagem[UTL_TAM_MAX_P];
11      float totalEmKb;
12      float usadoEmKb;
13
14      struct agtListaDeParticoes_ *proximo;
15  } agtListaDeParticoes;
16
17  bool agtLeParticoes(agtListaDeParticoes *agtLParticoes);
18  void agtRemoverListaDeParticoes(agtListaDeParticoes *agtLParticoes);
19
20  #include "agtParticoes.c"
21
22  #endif
```

- *agtParticoes.c*: possui funções para manipular a lista de partições e para extrair informações referentes as partições.

```
1   bool agtLeParticoes(agtListaDeParticoes *agtLParticoes)
2   {
3       struct mntent *mnt;
4       char *table = MOUNTED;
5       FILE *fp;
6       int x = 1;
7
8       fp = setmntent (table, "r");
9       if (fp == NULL)
10      {
11        return false;
12      }
13
14      while ((mnt = getmntent(fp)))
15      {
16          struct statfs fsd;
17
18          if (statfs(mnt->mnt_dir, &fsd) < 0)
19          {
20                return false;
21          }
22          if (x > 1)
23          {
```

```
24              agtLParticoes->proximo = (agtListaDeParticoes*)
    malloc(sizeof(agtListaDeParticoes));
25              agtLParticoes = agtLParticoes->proximo;
26          }
27
28          agtLParticoes->proximo = (agtListaDeParticoes*) NULL;
29          strcpy(agtLParticoes->nome, mnt->mnt_fsname);
30          strcpy(agtLParticoes->pontoDeMontagem, mnt->mnt_dir);
31          agtLParticoes->totalEmKb = (float)fsd.f_blocks * (fsd.f_bsize / 1024.0);
32          agtLParticoes->usadoEmKb = (float)(fsd.f_blocks - fsd.f_bfree)  *
    (fsd.f_bsize / 1024.0);
33
34          x++;
35      }
36      return true;
37  }
38
39  void agtRemoverListaDeParticoes(agtListaDeParticoes *agtLParticoes)
40  {
41      agtListaDeParticoes *auxiliar = (agtListaDeParticoes*) NULL;
42      agtListaDeParticoes *anterior = (agtListaDeParticoes*) NULL;
43      int quantidade = 0;
44
45      for (auxiliar = agtLParticoes; auxiliar != (agtListaDeParticoes*) NULL;  )
46      {
47          anterior = auxiliar;
48          auxiliar = auxiliar->proximo;
49          anterior->proximo = (agtListaDeParticoes*) NULL;
50          free(anterior);
51          quantidade++;
52      }
53
54      utlDebug("%d particoes removidas da lista.", UTL_NOTICIA, quantidade);
55  }
```

- *agtTabelaDeParticoes_.h*: define funções encontradas no arquivo *agtTabelaDeParticoes_.c*.

```
1   /* Ao gerar novamente a mib deve-se:
2   *  -> em tccPartitonTable:
3   *       s table_set -> a_table_set
4   *       + netsnmp_table_data_set *a_table_set;
5   *       + #include "agtTabelaDeParticoes_.h"
6   *       - void initialize_table_tccPartitionTable() :: netsnmp_table_data_set
    *a_table_set;
7   *       + int tccPartitonTable_handler() :: return
    agtTabelaDeParticoesHandler(handler, reginfo, reqinfo, requests);
8   */
9
10  #ifndef AGTTABELADEPARTICOES1_H
11  #define AGTTABELADEPARTICOES1_H
12
13  int aIDDaLinha = 0;
14
15  int agtTabelaDeParticoesHandler( netsnmp_mib_handler *handler,
    netsnmp_handler_registration *reginfo, netsnmp_agent_request_info *reqinfo,
    netsnmp_request_info *requests);
```

```
16 void agtAdicionaParticoesNaMIB(char *dataHoraDaCaptura, char *nome, char
   *pontoDeMontagem, char *totalEmKb, char *usadoEmKb);
17
18 #include "agtTabelaDeParticoes_.c"
19
20 #endif
```

- *agtTabelaDeParticoes_.c*: possui funções para incluir e remover linhas da tabela *tccPartitionTable.* A função para excluir uma linha é executada após o envio da informação para o coletor.

```
1  int agtTabelaDeParticoesHandler( netsnmp_mib_handler *handler,
   netsnmp_handler_registration *reginfo, netsnmp_agent_request_info *reqinfo,
   netsnmp_request_info *requests)
2  {
3      if (reqinfo->mode == MODE_GET)
4      {
5          char buf[40];
6          snprint_objid(buf, 40-1, requests->requestvb->name,requests->requestvb-
   >name_length);
7
8          char nomeDaColuna[40];
9          char numeroDaLinha[40];
10         utlCortaTexto(nomeDaColuna, buf, ".", 0);
11         utlCortaTexto(numeroDaLinha, buf, ".", 1);
12
13
14         if (!strcmp("TCC-MIB::aUsadoEmKb", nomeDaColuna))
15         {
16             netsnmp_table_row *row;
17             row=netsnmp_extract_table_row (requests);
18
19             netsnmp_table_dataset_remove_and_delete_row (a_table_set, row);
20             if (atoi(numeroDaLinha) == aIDDaLinha)
21             {
22                 aIDDaLinha = 0;
23             }
24         }
25     }
26
27     return SNMP_ERR_NOERROR;
28 }
29
30 void agtAdicionaParticoesNaMIB(char *dataHoraDaCaptura, char *nome, char
   *pontoDeMontagem, char *totalEmKb, char *usadoEmKb)
31 {
32     netsnmp_table_row *row;
33     aIDDaLinha++;
34
35     row = netsnmp_create_table_data_row();
36
37     netsnmp_table_row_add_index(row, ASN_INTEGER, &aIDDaLinha,
38                                 sizeof(aIDDaLinha));
39     netsnmp_set_row_column(row, COLUMN_AIDDALINHA, ASN_INTEGER,
```

```
40                          &aIDDaLinha, sizeof(aIDDaLinha));
41      netsnmp_mark_row_column_writable(row, COLUMN_AIDDALINHA, 1);
42
43      netsnmp_set_row_column(row, COLUMN_ADATAHORADACOLETA, ASN_OCTET_STR,
44                             dataHoraDaCaptura, strlen(dataHoraDaCaptura));
45      netsnmp_mark_row_column_writable(row, COLUMN_ADATAHORADACOLETA, 1);
46
47      netsnmp_set_row_column(row, COLUMN_ANOME, ASN_OCTET_STR,
48                             nome, strlen(nome));
49      netsnmp_mark_row_column_writable(row, COLUMN_ANOME, 1);
50
51      netsnmp_set_row_column(row, COLUMN_APONTODEMONTAGEM, ASN_OCTET_STR,
52                             pontoDeMontagem, strlen(pontoDeMontagem));
53      netsnmp_mark_row_column_writable(row, COLUMN_APONTODEMONTAGEM, 1);
54
55      netsnmp_set_row_column(row, COLUMN_ATOTALEMKB, ASN_OCTET_STR,
56                             totalEmKb, strlen(totalEmKb));
57      netsnmp_mark_row_column_writable(row, COLUMN_ATOTALEMKB, 1);
58
59      netsnmp_set_row_column(row, COLUMN_AUSADOEMKB, ASN_OCTET_STR,
60                             usadoEmKb, strlen(usadoEmKb));
61      netsnmp_mark_row_column_writable(row, COLUMN_AUSADOEMKB, 1);
62
63      netsnmp_table_dataset_add_row(a_table_set, row);
64
65  }
```

- *agtTabelaDeProcessos_.h*: define funções encontradas no arquivo *agtTabelaDeProcessos_.c.*

```
1   /* Ao gerar novamente a mib deve-se:
2   *  -> em tccProcessTable:
3   *       + netsnmp_table_data_set *table_set;
4   *       + #include "agtTabelaDeProcessos_.h"
5   *       - void initialize_table_tccProcessTable() :: netsnmp_table_data_set
    *table_set;
6   *       + int tccProcessTable_handler() :: return
    agtTabelaDeProcessosHandler(handler, reginfo, reqinfo, requests);
7   */
8
9   #ifndef AGTTABELADEPROCESSOS1_H
10  #define AGTTABELADEPROCESSOS1_H
11
12  int IDDaLinha = 0;
13
14  int agtTabelaDeProcessosHandler( netsnmp_mib_handler *handler,
    netsnmp_handler_registration *reginfo, netsnmp_agent_request_info *reqinfo,
    netsnmp_request_info *requests);
15  void agtAdicionaProcessoNaMIB(int pid, char *dataHoraDaCaptura, char *comm, char
    *usuario, char *percentualCPU, char *percentualMemoria);
16
17  #include "agtTabelaDeProcessos_.c"
18
19  #endif
```

- *agtTabelaDeProcessos_.c*: possui funções para incluir e remover linhas da tabela *tccProcessTable.* A função para excluir uma linha é executada após o envio da informação para o coletor.

```
1   int agtTabelaDeProcessosHandler( netsnmp_mib_handler *handler,
    netsnmp_handler_registration *reginfo, netsnmp_agent_request_info *reqinfo,
    netsnmp_request_info *requests)
2   {
3       if (reqinfo->mode == MODE_GET)
4       {
5           char buf[40];
6           snprint_objid(buf, 40-1, requests->requestvb->name,requests->requestvb-
    >name_length);
7
8           char nomeDaColuna[40];
9           char numeroDaLinha[40];
10          utlCortaTexto(nomeDaColuna, buf, ".", 0);
11          utlCortaTexto(numeroDaLinha, buf, ".", 1);
12
13
14          if (!strcmp("TCC-MIB::pUtilizacaoDaMemoria", nomeDaColuna))
15          {
16              netsnmp_table_row *row;
17              row=netsnmp_extract_table_row (requests);
18
19              netsnmp_table_dataset_remove_and_delete_row (table_set, row);
20              if (atoi(numeroDaLinha) == IDDaLinha)
21              {
22                  IDDaLinha = 0;
23              }
24          }
25      }
26
27      return SNMP_ERR_NOERROR;
28  }
29
30  void agtAdicionaProcessoNaMIB(int pid, char *dataHoraDaCaptura, char *comm, char
    *usuario, char *percentualCPU, char *percentualMemoria)
31  {
32      netsnmp_table_row *row;
33      IDDaLinha++;
34
35      row = netsnmp_create_table_data_row();
36
37      netsnmp_table_row_add_index(row, ASN_INTEGER, &IDDaLinha,
38                                  sizeof(IDDaLinha));
39      netsnmp_set_row_column(row, COLUMN_PIDDALINHA, ASN_INTEGER,
40                             &IDDaLinha, sizeof(IDDaLinha));
41      netsnmp_mark_row_column_writable(row, COLUMN_PIDDALINHA, 1);
42
43
44      netsnmp_set_row_column(row, COLUMN_PID, ASN_INTEGER,
45                             &pid, sizeof(pid));
46      netsnmp_mark_row_column_writable(row, COLUMN_PID, 1);
47
48      netsnmp_set_row_column(row, COLUMN_PDATAHORADACOLETA, ASN_OCTET_STR,
49                               dataHoraDaCaptura, strlen(dataHoraDaCaptura));
50      netsnmp_mark_row_column_writable(row, COLUMN_PDATAHORADACOLETA, 1);
51
52      netsnmp_set_row_column(row, COLUMN_PNOME, ASN_OCTET_STR,
```

```
53                              comm, strlen(comm));
54      netsnmp_mark_row_column_writable(row, COLUMN_PNOME, 1);
55
56      netsnmp_set_row_column(row, COLUMN_PUSUARIO, ASN_OCTET_STR, usuario,
57                             strlen(usuario));
58      netsnmp_mark_row_column_writable(row, COLUMN_PUSUARIO, 1);
59
60      netsnmp_set_row_column(row, COLUMN_PUTILIZACAODACPU, ASN_OCTET_STR,
    percentualCPU,
61                             strlen(percentualCPU));
62      netsnmp_mark_row_column_writable(row, COLUMN_PUTILIZACAODACPU, 1);
63
64      netsnmp_set_row_column(row, COLUMN_PUTILIZACAODAMEMORIA, ASN_OCTET_STR,
    percentualMemoria,
65                             strlen(percentualMemoria));
66      netsnmp_mark_row_column_writable(row, COLUMN_PUTILIZACAODAMEMORIA, 1);
67
68      netsnmp_table_dataset_add_row(table_set, row);
69
70  }
```

- *agtTabelaDeTemperatura_.h*: define funções encontradas no arquivo *agtTabelaDeTemperatura_.c.*

```
1   /* Ao gerar novamente a mib deve-se:
2   *  -> em tccTemperatureTable:
3   *       s table_set -> t_table_set
4   *       + netsnmp_table_data_set *t_table_set;
5   *       + #include "agtTabelaDeTemperatura_.h"
6   *       - void initialize_table_tccTemperatureTable() :: netsnmp_table_data_set
    *t_table_set;
7   *       + int tccTemperatureTable_handler() :: return
    agtTabelaDeTemperaturaHandler(handler, reginfo, reqinfo, requests);
8   */
9
10  #ifndef AGTTABELADETEMPERATURA1_H
11  #define AGTTABELADETEMPERATURA1_H
12
13  int tIDDaLinha = 0;
14
15  int agtTabelaDeTemperaturaHandler( netsnmp_mib_handler *handler,
    netsnmp_handler_registration *reginfo, netsnmp_agent_request_info *reqinfo,
    netsnmp_request_info *requests);
16  void agtAdicionaTemperaturaNaMIB(char *dataHoraDaCaptura, int temperatura);
17
18  #include "agtTabelaDeTemperatura_.c"
19
20  #endif
```

- *agtTabelaDeTemperatura_.c*: possui funções para incluir e remover linhas da tabela *tccTemperatureTable.* A função para excluir uma linha é executada após o envio da informação para o coletor.

```
1   int agtTabelaDeTemperaturaHandler( netsnmp_mib_handler *handler,
    netsnmp_handler_registration *reginfo, netsnmp_agent_request_info *reqinfo,
    netsnmp_request_info *requests)
2   {
3       if (reqinfo->mode == MODE_GET)
4       {
5           char buf[40];
6           snprint_objid(buf, 40-1, requests->requestvb->name,requests->requestvb-
    >name_length);
7
8           char nomeDaColuna[40];
9           char numeroDaLinha[40];
10          utlCortaTexto(nomeDaColuna, buf, ".", 0);
11          utlCortaTexto(numeroDaLinha, buf, ".", 1);
12
13
14          if (!strcmp("TCC-MIB::tTemperatura", nomeDaColuna))
15          {
16              netsnmp_table_row *row;
17              row=netsnmp_extract_table_row (requests);
18
19              netsnmp_table_dataset_remove_and_delete_row (t_table_set, row);
20              if (atoi(numeroDaLinha) == tIDDaLinha)
21              {
22                  tIDDaLinha = 0;
23              }
24          }
25      }
26
27      return SNMP_ERR_NOERROR;
28  }
29
30  void agtAdicionaTemperaturaNaMIB(char *dataHoraDaCaptura, int temperatura)
31  {
32      netsnmp_table_row *row;
33      tIDDaLinha++;
34
35      row = netsnmp_create_table_data_row();
36
37      netsnmp_table_row_add_index(row, ASN_INTEGER, &tIDDaLinha,
38                                  sizeof(tIDDaLinha));
39      netsnmp_set_row_column(row, COLUMN_TIDDALINHA, ASN_INTEGER,
40                             &tIDDaLinha, sizeof(tIDDaLinha));
41      netsnmp_mark_row_column_writable(row, COLUMN_TIDDALINHA, 1);
42
43      netsnmp_set_row_column(row, COLUMN_TDATAHORADACOLETA, ASN_OCTET_STR,
44                             dataHoraDaCaptura, strlen(dataHoraDaCaptura));
45      netsnmp_mark_row_column_writable(row, COLUMN_TDATAHORADACOLETA, 1);
46
47      netsnmp_set_row_column(row, COLUMN_TTEMPERATURA, ASN_INTEGER,
48                             &temperatura, sizeof(temperatura));
49      netsnmp_mark_row_column_writable(row, COLUMN_TTEMPERATURA, 1);
50
51      netsnmp_table_dataset_add_row(t_table_set, row);
52
53  }
```

APÊNDICE C Código fonte do coletor

Neste apêndice pode-se visualizar cada arquivo fonte da aplicação coletord. O coletor é composto por 14 arquivos fontes, onde 12 são mostrados abaixo e 2 (util.h e util.c) são descritos no apêndice 4.

- *coletord.c*: fonte principal que requisita as informações ao agentes, agrupa as informações de processos e armazena no banco de dados.

```
1   #include "coletord.h"
2   #include <libpq-fe.h>
3
4   int main(int argc, char **argv)
5   {
6       cltConfiguracao *cltConf;
7       cltConf = (cltConfiguracao*) malloc(sizeof(cltConfiguracao));
8       //Busca as configurações do coletor
9       cfg_opt_t opts[] = {
10                  CFG_SIMPLE_STR ("snmpComunidade", &cltConf->snmpComunidade),
11                  CFG_SIMPLE_INT ("snmpPorta", &cltConf->snmpPorta),
12                  CFG_SIMPLE_STR ("bdHost", &cltConf->bdHost),
13                  CFG_SIMPLE_STR ("bdPorta", &cltConf->bdPorta),
14                  CFG_SIMPLE_STR ("bdUsuario", &cltConf->bdUsuario),
15                  CFG_SIMPLE_STR ("bdNome", &cltConf->bdNome),
16                  CFG_SIMPLE_STR ("bdPassword", &cltConf->bdPassword),
17                  CFG_SIMPLE_INT ("cltTempoDeCaptura", &cltConf-
    >cltTempoDeCaptura),
18                  CFG_SIMPLE_INT ("cltExibeNoticia", &cltConf->cltExibeNoticia),
19                  CFG_STR_LIST   ("cltIPsParaCaptura", NULL, CFGF_NONE),
20                  CFG_END()
21                  };
22      utlLeConfiguracao(CLT_ARQUIVO_CONFIGURACAO, opts);
23
24      UTL_EXIBE_NOTICIA = cltConf->cltExibeNoticia;
25      utlListaConfiguracao *cltIPs, *cltIPsInicio;
26      cltIPs = (utlListaConfiguracao*) malloc(sizeof(utlListaConfiguracao));
27      utlLeListaConfiguracao(CLT_ARQUIVO_CONFIGURACAO, opts, "cltIPsParaCaptura",
    cltIPs);
28
29      utlDebug("Iniciando coletor.", UTL_NOTICIA);
30
31      //percore os ips a serem monitorados
32      cltIPsInicio = cltIPs;
33      while (1)
34      {
35          int tempoEmSegundos;
36          utlTempoEmSegundos(&tempoEmSegundos);
```

```
37          while (cltIPs != (utlListaConfiguracao*) NULL)
38          {
39              strcpy(cltConf->snmpHost, cltIPs->valor);
40              cltIPs = cltIPs->proximo;
41
42              strcpy(cltConf->cltNomeDoHost, cltIPs->valor);
43              cltIPs = cltIPs->proximo;
44
45              cltColetaDados(cltConf);
46          }
47          cltIPs = cltIPsInicio;
48          int tempoEmSegundos2;
49          utlTempoEmSegundos(&tempoEmSegundos2);
50          int tempoDiff = tempoEmSegundos2 - tempoEmSegundos;

51          sleep(cltConf->cltTempoDeCaptura-tempoDiff);
52      }
53
54      utlRemoveListaConfiguracao(cltIPs);
55      free(cltConf);
56
57      return 0;
58  }
59
60  void cltColetaDados(void *conf)
61  {
62      struct snmp_session sessao, *ss;
63      char *snmpErro;
64
65      cltConfiguracao *cltConf;
66      cltConf = conf;
67
68      utlDebug("Coletando dados de %s.", UTL_NOTICIA, cltConf->snmpHost);
69
70      ss = snmpCriarSessao(&sessao, cltConf);
71      if (ss == NULL)
72      {
73          snmp_error(&sessao, NULL, NULL, &snmpErro);
74          utlDebug("Erro ao criar a sessão snmp - %s\n", UTL_CUIDADO, snmpErro);
75      }
76
77      cltTabelaDeProcessos *cltTP;
78      cltTP     = (cltTabelaDeProcessos*) malloc(sizeof(cltTabelaDeProcessos));
79
80      if (cltBuscaTabelaDeProcessos(cltTP, ss))
81      {
82          cltTabelaDeProcessos *cltNovaTP;
83          cltNovaTP = malloc(sizeof(cltTabelaDeProcessos));
84          cltNovaEstruturaAgrupada(cltTP, cltNovaTP);
85          cltGravarTabelaDeProcessos(cltNovaTP, cltConf);
86          cltRemoverTabelaDeProcessos(cltNovaTP);
87      }
88      cltRemoverTabelaDeProcessos(cltTP);
89
90
91      cltTabelaDeTemperatura *cltTT;
92      cltTT     = (cltTabelaDeTemperatura*) malloc(sizeof(cltTabelaDeTemperatura));
93      if (cltBuscaTabelaDeTemperatura(cltTT, ss))
94      {
95          cltGravarTabelaDeTemperatura(cltTT, cltConf);
96      }
97      cltRemoverTabelaDeTemperatura(cltTT);
98
99      cltTabelaDeParticoes *cltTPa;
100     cltTPa     = (cltTabelaDeParticoes*) malloc(sizeof(cltTabelaDeParticoes));
101     if (cltBuscaTabelaDeParticoes(cltTPa, ss))
102     {
```

```
103         cltGravarTabelaDeParticoes(cltTPa, cltConf);
104     }
105     cltRemoverTabelaDeParticoes(cltTPa);
106
107
108
109     snmpFecharSessao(ss);
110 }
```

- *coletord.h*: define funções e estruturas encontradas no arquivo coletord.*c*.

```
1   #ifndef COLETOR_H
2   #define COLETOR_H
3
4   #include "util.h"
5   #include <pthread.h>
6
7   //Define o nome do arquivo de configuração do coletor.
8   #define CLT_ARQUIVO_CONFIGURACAO "coletord.conf"
9
10  //Estrutura que armazena os dados de configuração do coletor
11  typedef struct cltConfiguracao_
12  {
13      char *snmpComunidade;
14      char snmpHost[UTL_TAM_MAX_BP];
15      int  snmpPorta;
16      char *bdHost;
17      char *bdPorta;
18      char *bdUsuario;
19      char *bdNome;
20      char *bdPassword;
21      int  cltTempoDeCaptura;
22      int  cltExibeNoticia;
23      char cltNomeDoHost[UTL_TAM_MAX_P]; //utilizado para gravar o nome da máquina
24
25  } cltConfiguracao;
26
27  typedef struct cltThreads_
28  {
29      pthread_t thread;
30      struct cltThreads_ *proximo;
31  } cltThreads;
32
33  void cltColetaDados(void *conf);
34
35  #include "cltBaseDeDados.h"
36  #include "cltSnmp.h"
37  #include "cltTabelaDeProcessos.h"
38  #include "cltTabelaDeTemperatura.h"
39  #include "cltTabelaDeParticoes.h"
40
41  #endif
```

- *cltBaseDeDados.h*: define funções e estruturas encontradas no arquivo cltBaseDeDados.*c*.

```
1   #ifndef CLTBASEDEDADOS_H
2   #define CLTBASEDEDADOS_H
3
4   #include <stdio.h>
5   #include <libpq-fe.h>
6
7   PGconn *bdConectar(cltConfiguracao *cltConf);
8   void bdExecutar(PGconn *conexao, char *sql);
9   void bdFechar(PGconn *conexao);
10
11  #include "util.h"
12  #include "cltBaseDeDados.c"
13
14  #endif
```

- *cltBaseDeDados.c*: funções para acesso ao banco de dados *postgresql*.

```
1   PGconn *bdConectar(cltConfiguracao *cltConf)
2   {
3       char bdParam[UTL_TAM_MAX_M] = "";
4       PGconn *conexao = NULL;
5
6       //Monta o parametro para conexão
7       if (cltConf->bdHost)
8       {
9           strcat(bdParam, " host=");
10          strcat(bdParam, cltConf->bdHost);
11      }
12      if (cltConf->bdPorta)
13      {
14          strcat(bdParam, " port=");
15          strcat(bdParam, cltConf->bdPorta);
16      }
17      if (cltConf->bdUsuario)
18      {
19          strcat(bdParam, " user=");
20          strcat(bdParam, cltConf->bdUsuario);
21      }
22      if (cltConf->bdNome)
23      {
24          strcat(bdParam, " dbname=");
25          strcat(bdParam, cltConf->bdNome);
26      }
27      if (cltConf->bdPassword)
28      {
29          strcat(bdParam, " password=");
30          strcat(bdParam, cltConf->bdPassword);
31      }
32      conexao = PQconnectdb(bdParam);
33      if(PQstatus(conexao) != CONNECTION_OK)
34      {
```

```
35          utlDebug("Falha na conexão. %s", UTL_ERRO_FATAL,
    PQerrorMessage(conexao));
36          PQfinish(conexao);
37      }
38
39      return conexao;
40  }
41
42  void bdExecutar(PGconn *conexao, char *sql)
43  {
44      PGresult *resultado;
45
46      resultado = PQexec(conexao, sql);
47
48      switch(PQresultStatus(resultado))
49      {
50          case PGRES_EMPTY_QUERY:
51              utlDebug("O sql está vazio.", UTL_CUIDADO);
52              break;
53          case PGRES_FATAL_ERROR:
54              utlDebug("Erro no sql %s", UTL_CUIDADO,
    PQresultErrorMessage(resultado));
55              break;
56          case PGRES_COMMAND_OK:
57              break;
58          default:
59              utlDebug("Problema não identificado ao executar o sql.",
    UTL_CUIDADO);
60              break;
61      }
62
63      if (resultado)
64      {
65          PQclear(resultado);
66      }
67  }
68
69  void bdFechar(PGconn *conexao)
70  {
71      if(conexao != NULL)
72      {
73          PQfinish(conexao);
74      }
75  }
```

- cltSnmp.h: define funções e estruturas encontradas no arquivo cltSnmp.*c*.

```
1   #ifndef CLTSNMP_H
2   #define CLTSNMP_H
3
4   #include <stdio.h>
5   #include <net-snmp/net-snmp-config.h>
6   #include <net-snmp/net-snmp-includes.h>
7
8   struct snmp_session *snmpCriarSessao(struct snmp_session *sessao, cltConfiguracao
    *cltConf);
9   void snmpFecharSessao(struct snmp_session *sessao);
10  bool snmpGet(struct snmp_session *ss, char *objeto, void *resultado, int
    *maisLinhas);
```

```
11
12 //snmpWal foi utilizado temporariamente
13 int snmpWalk(char *objeto, cltConfiguracao *cltConf);
14
15 #include "util.h"
16 #include "cltSnmp.c"
17
18 #endif
```

- *cltSnmp.c*: função para abrir uma sessão com o agente e requisitar informações de um objeto.

```
1  struct snmp_session *snmpCriarSessao(struct snmp_session *sessao, cltConfiguracao
   *cltConf)
2  {
3      utlDebug("Criando uma sessão snmp.", UTL_NOTICIA);
4
5      snmp_sess_init(sessao);
6      sessao->version = SNMP_VERSION_2c;
7      sessao->peername = cltConf->snmpHost;
8      sessao->remote_port = cltConf->snmpPorta;
9      sessao->community = (unsigned char *)cltConf->snmpComunidade;
10     sessao->community_len = strlen(cltConf->snmpComunidade);
11
12     return(snmp_open(sessao));
13 }
14
15 void snmpFecharSessao(struct snmp_session *sessao)
16 {
17     if (sessao != NULL)
18     {
19         snmp_close(sessao);
20     }
21 }
22
23 bool snmpGet(struct snmp_session *ss, char *objeto, void *resultado, int
   *maisLinhas)
24 {
25     utlDebug("Caminhando pelo objeto '%s'.", UTL_NOTICIA, objeto);
26
27     *maisLinhas = 0;
28
29     struct snmp_pdu *pdu = NULL, *resposta = NULL;
30     struct variable_list *vars;
31
32     oid root[MAX_OID_LEN];
33     size_t rootLen = MAX_OID_LEN;
34
35     int terminou = 0;
36     int status = 0;
37     int valoresEncontrados = 0;
38
39
40     char temp[UTL_TAM_MAX_P], buf[UTL_TAM_MAX_P]; int count;
41
42     init_snmp("snmpGet");
43
```

```
44      if (!snmp_parse_oid(objeto, root, &rootLen))
45      {
46          printf("Parse\n");
47          return false;
48      }
49
50      pdu = snmp_pdu_create(SNMP_MSG_GET);
51      snmp_add_null_var(pdu, root, rootLen);
52
53      status = snmp_synch_response(ss, pdu, &resposta);
54      if (status == STAT_SUCCESS)
55      {
56          if (resposta->errstat == SNMP_ERR_NOERROR)
57          {
58              for(vars = resposta->variables; vars; vars = vars->next_variable)
59              {
60                  valoresEncontrados++;
61                  snprint_variable(buf,sizeof(buf),vars->name,vars-
    >name_length,vars);
62                  strcat(buf,"\n");
63
64                  if (vars->type == ASN_INTEGER)
65                  {
66                      int *res = resultado;
67                      *res = *(vars->val.integer);
68
69                      *maisLinhas = 1;
70                  }
71                  else if (vars->type == ASN_OCTET_STR)
72                  {
73                      char *sp = (char *) malloc(1 + vars->val_len);
74                      memcpy(sp, vars->val.string, vars->val_len);
75                      sp[vars->val_len] = '\0';
76                      sprintf(resultado, "%s", sp);
77                      free(sp);
78                      *maisLinhas = 1;
79                  }
80              }
81          }
82          else
83          {
84              //Trata os erros de resposta
85              if (resposta->errstat == SNMP_ERR_NOSUCHNAME)
86              {
87                  printf("    END MIB\n");
88              }
89              else
90              {
91                  utlDebug("Erro no pacote: %s", UTL_CUIDADO,
    snmp_errstring(resposta->errstat));
92                  if (resposta->errstat == SNMP_ERR_NOSUCHNAME)
93                  {
94                      sprintf(temp,"A requisição deste identificador de objeto
    falhou: ");
95                      for(count = 1, vars = resposta->variables; vars && count !=
    resposta->errindex; vars = vars->next_variable, count++)
96                      {
97                          if (vars)
98                          {
99                              snprint_objid(buf,sizeof(buf),vars->name,vars-
    >name_length);
100                             strcat(temp,buf);
101                         }
102                     }
103                     printf("E: %s\n", temp);
104                 }
105             }
```

```
106         }
107     }
108     else if (status == STAT_TIMEOUT)
109     {
110         utlDebug("Tempo esgotou enquanto conectava em '%s'", UTL_CUIDADO, ss-
    >peername);
111         snmp_close(ss);
112         return false;
113     }
114     else
115     {    /* status == STAT_ERROR */
116         snmp_sess_perror("snmpGet", ss);
117         terminou = 1;
118     }
119     if (resposta)
120     {
121         snmp_free_pdu(resposta);
122     }
123
124     return true;
125 }
```

- *cltTabelaDeProcessos.h*: define funções e estruturas encontradas no arquivo *agtTabelaDeProcessos.c*.

```
1   #ifndef CLTTABELADEPROCESSOS_H
2   #define CLTTABELADEPROCESSOS_H
3
4   #include <stdio.h>
5
6   typedef struct cltTabelaDeProcessos_
7   {
8       int IDDaLinha;
9       int ID;
10      char dataHoraInicial[UTL_TAM_MAX_P];
11      char dataHoraFinal[UTL_TAM_MAX_P];
12      char nome[UTL_TAM_MAX_P];
13      char usuario[UTL_TAM_MAX_P];
14      float utilizacaoDaCPU;
15      float utilizacaoDaMemoria;
16      int quantidadeDeAgrupamentos; //utilizado para fazer a média da memória
17      struct cltTabelaDeProcessos_ *proximo;
18  } cltTabelaDeProcessos;
19
20  bool cltBuscaTabelaDeProcessos(cltTabelaDeProcessos *cltTP, struct snmp_session
    *ss);
21  void cltImprimeTabelaDeProcessos(cltTabelaDeProcessos *cltTP, char
    *cltNomeDoHost);
22  void cltNovaEstruturaAgrupada(cltTabelaDeProcessos *cltTP, cltTabelaDeProcessos
    *cltNovaTP);
23  void cltGravarTabelaDeProcessos(cltTabelaDeProcessos *cltTP, cltConfiguracao *
    cltConf);
24  void cltRemoverTabelaDeProcessos(cltTabelaDeProcessos *cltTP);
25
26  #include "util.h"
27  #include "cltBaseDeDados.h"
28  #include "cltTabelaDeProcessos.c"
```

```
29
30 #endif
```

- *cltTabelaDeProcessos.c*: funções para requisitar, agrupar, visualizar e armazenar as informações dos processos.

```
1  bool cltBuscaTabelaDeProcessos(cltTabelaDeProcessos *cltTP, struct snmp_session
   *ss)
2  {
3      int x = 1, acabou = 0, maisLinhas = 0;
4      char buf[50];
5      char resultadoTmp[50];
6      while(!acabou)
7      {
8          sprintf(buf, "TCC-MIB::pIDDaLinha.%d", x);
9          if (!snmpGet(ss, buf, &cltTP->IDDaLinha, &maisLinhas))
10         {
11             cltTP->proximo = (cltTabelaDeProcessos*) NULL;
12             cltTP = (cltTabelaDeProcessos*) NULL;
13             return false;
14         }
15
16         if (maisLinhas == 0)
17         {
18             if (x == 1)
19             {
20                 cltTP->proximo = (cltTabelaDeProcessos*) NULL;
21                 return false;
22             }
23             acabou = 1;
24             break;
25         }
26
27         if (x > 1)
28         {
29             cltTP->proximo = (cltTabelaDeProcessos*)
   malloc(sizeof(cltTabelaDeProcessos));
30             cltTP = cltTP->proximo;
31         }
32         sprintf(buf, "TCC-MIB::pID.%d", x);
33         snmpGet(ss, buf, &cltTP->ID, &maisLinhas);
34
35         sprintf(buf, "TCC-MIB::pDataHoraDaColeta.%d", x);
36         snmpGet(ss, buf, &cltTP->dataHoraInicial, &maisLinhas);
37         strcpy(cltTP->dataHoraFinal, cltTP->dataHoraInicial);
38
39         sprintf(buf, "TCC-MIB::pNome.%d", x);
40         snmpGet(ss, buf, &cltTP->nome, &maisLinhas);
41
42         sprintf(buf, "TCC-MIB::pUsuario.%d", x);
43         snmpGet(ss, buf, &cltTP->usuario, &maisLinhas);
44
45         sprintf(buf, "TCC-MIB::pUtilizacaoDaCPU.%d", x);
46         snmpGet(ss, buf, &resultadoTmp, &maisLinhas);
47         cltTP->utilizacaoDaCPU = atof(resultadoTmp);
48
49         sprintf(buf, "TCC-MIB::pUtilizacaoDaMemoria.%d", x);
```

```
50          snmpGet(ss, buf, &resultadoTmp, &maisLinhas);
51          cltTP->utilizacaoDaMemoria = atof(resultadoTmp);
52
53          cltTP->quantidadeDeAgrupamentos = 1;
54
55          cltTP->proximo = (cltTabelaDeProcessos*) NULL;
56          x++;
57      }
58
59      utlDebug("%d processo(s) adicinado(s) na lista.", UTL_NOTICIA, x-1);
60  printf("%d processo(s) adicinado(s) na lista.\n",  x-1);
61      return true;
62  }
63
64  void cltImprimeTabelaDeProcessos(cltTabelaDeProcessos *cltTP, char
    *cltNomeDoHost)
65  {
66      cltTabelaDeProcessos *auxiliar;
67      printf("Tabela de processos:\n");
68      for (auxiliar = cltTP; auxiliar != (cltTabelaDeProcessos*) NULL; auxiliar =
    auxiliar->proximo)
69      {
70          printf("%s %d %s %s %s %s %f %f\n", cltNomeDoHost,
71                                              auxiliar->ID,
72                                              auxiliar->dataHoraInicial,
73                                              auxiliar->dataHoraFinal,
74                                              auxiliar->nome,
75                                              auxiliar->usuario,
76                                              auxiliar->utilizacaoDaCPU,
77                                              auxiliar->utilizacaoDaMemoria);
78      }
79  }
80
81  void cltNovaEstruturaAgrupada(cltTabelaDeProcessos *cltTP, cltTabelaDeProcessos
    *cltNovaTP)
82  {
83      cltTabelaDeProcessos *auxiliar;
84      cltTabelaDeProcessos *auxiliarNova;
85      cltTabelaDeProcessos *inicioNova;
86      int x = 0;
87      int quantidadeDeTempos = 0; //Utilizado para fazer a média da utilização da
    CPU
88      char dataHoraAuxiliar[UTL_TAM_MAX_P] = "";
89      char dataHoraInicial[UTL_TAM_MAX_P] = "";
90      char dataHoraFinal[UTL_TAM_MAX_P] = "";
91
92      inicioNova = cltNovaTP;
93
94      strcpy(dataHoraInicial, cltTP->dataHoraInicial);
95      for (auxiliar = cltTP; auxiliar != (cltTabelaDeProcessos*) NULL; auxiliar =
    auxiliar->proximo)
96      {
97          int insereNova = 1;
98
99          if (strcmp(auxiliar->dataHoraInicial, dataHoraAuxiliar) != 0)
100         {
101             strcpy(dataHoraAuxiliar, auxiliar->dataHoraInicial);
102             quantidadeDeTempos++;
103         }
104
105         if (x > 0)
106         {
107             //Percore a nova estrutura para somar os valores de processos iguais
108             for (auxiliarNova = inicioNova; auxiliarNova !=
    (cltTabelaDeProcessos*) NULL; auxiliarNova = auxiliarNova->proximo)
109             {
110                 if (auxiliarNova->ID == auxiliar->ID && strcmp(auxiliarNova-
```

```
    >nome, auxiliar->nome) == 0 && strcmp(auxiliarNova->usuario, auxiliar->usuario)
    == 0)
111                 {
112                     auxiliarNova->utilizacaoDaCPU += auxiliar->utilizacaoDaCPU;
113                     auxiliarNova->utilizacaoDaMemoria += auxiliar-
    >utilizacaoDaMemoria;
114                     auxiliarNova->quantidadeDeAgrupamentos++;
115                     insereNova = 0;
116                 }
117             }
118         }
119 
120         if (insereNova)
121         {
122             if (x > 0 )
123             {
124                 cltNovaTP->proximo = (cltTabelaDeProcessos*)
    malloc(sizeof(cltTabelaDeProcessos));
125                 cltNovaTP = cltNovaTP->proximo;
126             }
127             //Adiciona novo item na lista
128             cltNovaTP->ID = auxiliar->ID;
129             strcpy(cltNovaTP->nome, auxiliar->nome);
130             strcpy(cltNovaTP->usuario, auxiliar->usuario);
131             cltNovaTP->utilizacaoDaCPU = auxiliar->utilizacaoDaCPU;
132             cltNovaTP->utilizacaoDaMemoria = auxiliar->utilizacaoDaMemoria;
133             cltNovaTP->quantidadeDeAgrupamentos = cltTP-
    >quantidadeDeAgrupamentos;
134 
135             cltNovaTP->proximo = (cltTabelaDeProcessos*) NULL;
136         }
137         strcpy(dataHoraFinal, auxiliar->dataHoraFinal);
138         x++;
139     }
140 
141     //Percore a nova estrutura para fazer a média dos valores
142     for (auxiliarNova = inicioNova; auxiliarNova != (cltTabelaDeProcessos*)
    NULL; auxiliarNova = auxiliarNova->proximo)
143     {
144         auxiliarNova->utilizacaoDaMemoria /= auxiliarNova-
    >quantidadeDeAgrupamentos;
145 
146         auxiliarNova->utilizacaoDaCPU /= quantidadeDeTempos;
147 
148         //Adiciona a data e hora inicial do agrupamento
149         strcpy(auxiliarNova->dataHoraInicial, dataHoraInicial);
150         //Adiciona a data e hora final do agrupamento
151         strcpy(auxiliarNova->dataHoraFinal, dataHoraFinal);
152     }
153 }
154 
155 void cltGravarTabelaDeProcessos(cltTabelaDeProcessos *cltTP, cltConfiguracao *
    cltConf)
156 {
157     cltTabelaDeProcessos *auxiliar;
158     char bufSql[UTL_TAM_MAX_G];
159     int quantidade = 0;
160 
161     PGconn *con = NULL;
162     con = bdConectar(cltConf);
163     for (auxiliar = cltTP; auxiliar != (cltTabelaDeProcessos*) NULL;  auxiliar =
    auxiliar->proximo)
164     {
165         sprintf(bufSql, "INSERT INTO processos (nomeDoHost, pid,
    datahorainicial, datahorafinal, nome, usuario, utilizacaoDaCPU,
    utilizacaoDaMemoria)"
166                                         " VALUES ('%s', %d,  '%s', '%s', '%s',
```

```
    '%s', %f, %f);",
167                                                 cltConf->cltNomeDoHost,
168                                                 auxiliar->ID,
169                                                 auxiliar->dataHoraInicial,
170                                                 auxiliar->dataHoraFinal,
171                                                 auxiliar->nome,
172                                                 auxiliar->usuario,
173                                                 auxiliar->utilizacaoDaCPU,
174                                                 auxiliar->utilizacaoDaMemoria);
175
176         bdExecutar(con, bufSql);
177         quantidade++;
178     }
179     bdFechar(con);
180
181     utlDebug("%d processo(s) gravado(s) na base.", UTL_NOTICIA, quantidade);
182 }
183
184 void cltRemoverTabelaDeProcessos(cltTabelaDeProcessos *cltTP)
185 {
186     cltTabelaDeProcessos *auxiliar = (cltTabelaDeProcessos*) NULL;
187     cltTabelaDeProcessos *anterior = (cltTabelaDeProcessos*) NULL;
188     int quantidade = 0;
189     for (auxiliar = cltTP; auxiliar != (cltTabelaDeProcessos*) NULL;  )
190     {
191         anterior = auxiliar;
192         auxiliar = auxiliar->proximo;
193         anterior->proximo = (cltTabelaDeProcessos*) NULL;
194         free(anterior);
195         quantidade++;
196     }
197
198     utlDebug("%d processo(s) removido(s) da lista.", UTL_NOTICIA, quantidade);
199 }
```

- cltTabelaDeTemperatura.h: define funções e estruturas encontradas no arquivo cltTabelaDeTemperatura.*c*.

```
1   #ifndef CLTTABELADETEMPERATURA_H
2   #define CLTTABELADETEMPERATURA_H
3
4   #include <stdio.h>
5
6   typedef struct cltTabelaDeTemperatura_
7   {
8       int IDDaLinha;
9       char dataHoraDaColeta[UTL_TAM_MAX_P];
10      int temperatura;
11      struct cltTabelaDeTemperatura_ *proximo;
12  } cltTabelaDeTemperatura;
13
14  bool cltBuscaTabelaDeTemperatura(cltTabelaDeTemperatura *cltTT, struct
    snmp_session *ss);
15  void cltImprimeTabelaDeTemperatura(cltTabelaDeTemperatura *cltTT, char
    *cltNomeDoHost);
16  void cltGravarTabelaDeTemperatura(cltTabelaDeTemperatura *cltTT, cltConfiguracao
    * cltConf);
```

```
17 void cltRemoverTabelaDeTemperatura(cltTabelaDeTemperatura *cltTT);
18
19 #include "util.h"
20 #include "cltBaseDeDados.h"
21 #include "cltTabelaDeTemperatura.c"
22
23 #endif
```

- *cltTabelaDeTemperatura.c*: funções para requisitar, visualizar e armazenar as informações da temperatura do processador.

```
1  bool cltBuscaTabelaDeTemperatura(cltTabelaDeTemperatura *cltTT, struct
   snmp_session *ss)
2  {
3      int x = 1, acabou = 0, maisLinhas = 0;
4      char buf[50];
5      while(!acabou)
6      {
7          sprintf(buf, "TCC-MIB::tIDDaLinha.%d", x);
8          if (!snmpGet(ss, buf, &cltTT->IDDaLinha, &maisLinhas))
9          {
10             cltTT->proximo = (cltTabelaDeTemperatura*) NULL;
11             cltTT = (cltTabelaDeTemperatura*) NULL;
12             return false;
13         }
14
15         if (maisLinhas == 0)
16         {
17             if (x == 1)
18             {
19                 cltTT->proximo = (cltTabelaDeTemperatura*) NULL;
20                 return false;
21             }
22             acabou = 1;
23             break;
24         }
25
26         if (x > 1)
27         {
28             cltTT->proximo = (cltTabelaDeTemperatura*)
   malloc(sizeof(cltTabelaDeTemperatura));
29             cltTT = cltTT->proximo;
30         }
31
32         sprintf(buf, "TCC-MIB::tDataHoraDaColeta.%d", x);
33         snmpGet(ss, buf, &cltTT->dataHoraDaColeta, &maisLinhas);
34
35         sprintf(buf, "TCC-MIB::tTemperatura.%d", x);
36         snmpGet(ss, buf, &cltTT->temperatura, &maisLinhas);
37
38         cltTT->proximo = (cltTabelaDeTemperatura*) NULL;
39         x++;
40     }
41
42     utlDebug("%d temperatura(s) adicinada(s) na lista.", UTL_NOTICIA, x-1);
43     return true;
44 }
```

```
45
46 void cltImprimeTabelaDeTemperatura(cltTabelaDeTemperatura *cltTT, char
   *cltNomeDoHost)
47 {
48     cltTabelaDeTemperatura *auxiliar;
49     printf("Tabela de temperatura:\n");
50     for (auxiliar = cltTT; auxiliar != (cltTabelaDeTemperatura*) NULL; auxiliar =
   auxiliar->proximo)
51     {
52         printf("%s %s %d\n", cltNomeDoHost,
53                              auxiliar->dataHoraDaColeta,
54                              auxiliar->temperatura);
55     }
56 }
57
58 void cltGravarTabelaDeTemperatura(cltTabelaDeTemperatura *cltTT, cltConfiguracao
   * cltConf)
59 {
60     cltTabelaDeTemperatura *auxiliar;
61     char bufSql[UTL_TAM_MAX_G];
62     int quantidade = 0;
63
64     PGconn *con = bdConectar(cltConf);
65     for (auxiliar = cltTT; auxiliar != (cltTabelaDeTemperatura*) NULL;  auxiliar
   = auxiliar->proximo)
66     {
67         sprintf(bufSql, "INSERT INTO temperaturas (nomeDoHost, datahoradacoleta,
   temperatura)"
68                                         " VALUES ('%s', '%s', %d);",
69                                                 cltConf->cltNomeDoHost,
70                                                 auxiliar->dataHoraDaColeta,
71                                                 auxiliar->temperatura);
72
73         bdExecutar(con, bufSql);
74         quantidade++;
75     }
76     bdFechar(con);
77
78     utlDebug("%d temperatura(s) gravada(s) na base.", UTL_NOTICIA, quantidade);
79 }
80
81 void cltRemoverTabelaDeTemperatura(cltTabelaDeTemperatura *cltTT)
82 {
83     cltTabelaDeTemperatura *auxiliar = (cltTabelaDeTemperatura*) NULL;
84     cltTabelaDeTemperatura *anterior = (cltTabelaDeTemperatura*) NULL;
85     int quantidade = 0;
86
87     for (auxiliar = cltTT; auxiliar != (cltTabelaDeTemperatura*) NULL;  )
88     {
89         anterior = auxiliar;
90         auxiliar = auxiliar->proximo;
91         anterior->proximo = (cltTabelaDeTemperatura*) NULL;
92         free(anterior);
93         quantidade++;
94     }
95
96     utlDebug("%d temperatura(s) removida(s) da lista.", UTL_NOTICIA, quantidade);
97 }
```

- cltTabelaDeParticoes.h: define funções e estruturas encontradas no arquivo cltTabelaDeParticoes*.c*.

```
1   #ifndef CLTTABELADEPARTICOES_H
2   #define CLTTABELADEPARTICOES_H
3
4   #include <stdio.h>
5
6   typedef struct cltTabelaDeParticoes_
7   {
8       int IDDaLinha;
9       char dataHoraDaColeta[UTL_TAM_MAX_P];
10      char nome[UTL_TAM_MAX_P];
11      char pontoDeMontagem[UTL_TAM_MAX_P];
12      float totalEmKb;
13      float usadoEmKb;
14
15      struct cltTabelaDeParticoes_ *proximo;
16  } cltTabelaDeParticoes;
17
18  bool cltBuscaTabelaDeParticoes(cltTabelaDeParticoes *cltTPa, struct snmp_session
    *ss);
19  void cltImprimeTabelaDeParticoes(cltTabelaDeParticoes *cltTPa, char
    *cltNomeDoHost);
20  void cltGravarTabelaDeParticoes(cltTabelaDeParticoes *cltTPa, cltConfiguracao *
    cltConf);
21  void cltRemoverTabelaDeParticoes(cltTabelaDeParticoes *cltTPa);
22
23  #include "util.h"
24  #include "cltBaseDeDados.h"
25  #include "cltTabelaDeParticoes.c"
26
27  #endif
```

- *cltTabelaDeParticoes.c*: funções para requisitar, visualizar e armazenar das partições.

```
1   bool cltBuscaTabelaDeParticoes(cltTabelaDeParticoes *cltTPa, struct snmp_session
    *ss)
2   {
3       int x = 1, acabou = 0, maisLinhas = 0;
4       char buf[50];
5       char resultadoTmp[50];
6       while(!acabou)
7       {
8           sprintf(buf, "TCC-MIB::aIDDaLinha.%d", x);
9           if (!snmpGet(ss, buf, &cltTPa->IDDaLinha, &maisLinhas))
10          {
11              cltTPa->proximo = (cltTabelaDeParticoes*) NULL;
12              cltTPa = (cltTabelaDeParticoes*) NULL;
13              return false;
14          }
```

```
15
16          if (maisLinhas == 0)
17          {
18              if (x == 1)
19              {
20                  cltTPa->proximo = (cltTabelaDeParticoes*) NULL;
21                  return false;
22              }
23              acabou = 1;
24              break;
25          }
26
27          if (x > 1)
28          {
29              cltTPa->proximo = (cltTabelaDeParticoes*)
    malloc(sizeof(cltTabelaDeParticoes));
30              cltTPa = cltTPa->proximo;
31          }
32
33          sprintf(buf, "TCC-MIB::aDataHoraDaColeta.%d", x);
34          snmpGet(ss, buf, &cltTPa->dataHoraDaColeta, &maisLinhas);
35
36          sprintf(buf, "TCC-MIB::aNome.%d", x);
37          snmpGet(ss, buf, &cltTPa->nome, &maisLinhas);
38
39          sprintf(buf, "TCC-MIB::aPontoDeMontagem.%d", x);
40          snmpGet(ss, buf, &cltTPa->pontoDeMontagem, &maisLinhas);
41
42          sprintf(buf, "TCC-MIB::aTotalEmKb.%d", x);
43          snmpGet(ss, buf, &resultadoTmp, &maisLinhas);
44          cltTPa->totalEmKb = atof(resultadoTmp);
45
46
47          sprintf(buf, "TCC-MIB::aUsadoEmKb.%d", x);
48          snmpGet(ss, buf, &resultadoTmp, &maisLinhas);
49          cltTPa->usadoEmKb = atof(resultadoTmp);
50
51          cltTPa->proximo = (cltTabelaDeParticoes*) NULL;
52          x++;
53      }
54
55      utlDebug("%d partição adicinada na lista.", UTL_NOTICIA, x-1);
56      return true;
57  }
58
59  void cltImprimeTabelaDeParticoes(cltTabelaDeParticoes *cltTPa, char
    *cltNomeDoHost)
60  {
61      cltTabelaDeParticoes *auxiliar;
62      printf("Tabela de particões:\n");
63      for (auxiliar = cltTPa; auxiliar != (cltTabelaDeParticoes*) NULL; auxiliar =
    auxiliar->proximo)
64      {
65          printf("%s %s %s %s %f %f\n", cltNomeDoHost,
66                                  auxiliar->dataHoraDaColeta,
67                                  auxiliar->nome,
68                                  auxiliar->pontoDeMontagem,
69                                  auxiliar->totalEmKb,
70                                  auxiliar->usadoEmKb);
71      }
72  }
73
74  void cltGravarTabelaDeParticoes(cltTabelaDeParticoes *cltTPa, cltConfiguracao *
    cltConf)
75  {
76      cltTabelaDeParticoes *auxiliar;
77      char bufSql[UTL_TAM_MAX_G];
```

```
78      int quantidade = 0;
79
80      PGconn *con = bdConectar(cltConf);
81      for (auxiliar = cltTPa; auxiliar != (cltTabelaDeParticoes*) NULL;  auxiliar =
    auxiliar->proximo)
82      {
83          sprintf(bufSql, "INSERT INTO particoes (nomeDoHost, datahoradacoleta,
    nome, pontoDeMontagem, totalEmKb, usadoEmKb)"
84                                          " VALUES ('%s', '%s', '%s', '%s', %f,
    %f);",
85                                                  cltConf->cltNomeDoHost,
86                                                  auxiliar->dataHoraDaColeta,
87                                                  auxiliar->nome,
88                                                  auxiliar->pontoDeMontagem,
89                                                  auxiliar->totalEmKb,
90                                                  auxiliar->usadoEmKb);
91
92          bdExecutar(con, bufSql);
93          quantidade++;
94      }
95      bdFechar(con);
96
97      utlDebug("%d partição(ões) gravada(s) na base.", UTL_NOTICIA, quantidade);
98  }
99
100 void cltRemoverTabelaDeParticoes(cltTabelaDeParticoes *cltTPa)
101 {
102     cltTabelaDeParticoes *auxiliar = (cltTabelaDeParticoes*) NULL;
103     cltTabelaDeParticoes *anterior = (cltTabelaDeParticoes*) NULL;
104     int quantidade = 0;
105     for (auxiliar = cltTPa; auxiliar != (cltTabelaDeParticoes*) NULL;  )
106     {
107         anterior = auxiliar;
108         auxiliar = auxiliar->proximo;
109         anterior->proximo = (cltTabelaDeParticoes*) NULL;
110         free(anterior);
111         quantidade++;
112     }
113
114     utlDebug("%d partição(ões) removida(s) da lista.", UTL_NOTICIA, quantidade);
115 }
```

APÊNDICE D Funções utilitárias

Os arquivos *util.h* e *util.c* formam criados com o intuito de conter funções úteis que possam ser utilizadas tanto pelo agente como pelo coletor. Como para coleta de temperatura não foi necessário a criação de estruturas de armazenamento temporário, criou-se somente uma função, na qual, foi escrita junto com as funções utilitárias.

- *util.h*: define funções e estruturas encontradas no arquivo *util.c*.

```
1   #ifndef UTIL_H
2   #define UTIL_H
3
4   //Tamanho máximo para uma string que armazena o valeor de uma configuração
5   #define UTL_TAM_MAX_VAL_CONF 50
6   #define UTL_TAM_MAX_BP 20
7   #define UTL_TAM_MAX_P 50
8   #define UTL_TAM_MAX_M 150
9   #define UTL_TAM_MAX_G 256
10
11  //Tipo de erros que o debug aceita
12  #define UTL_NOTICIA 1
13  #define UTL_CUIDADO 2
14  #define UTL_ERRO_FATAL 3
15
16  //Controle para exibição das notícias
17  int UTL_EXIBE_NOTICIA=0;
18
19  #define PROC_DIR "/proc"
20
21  #include <stdarg.h>
22  #include <stdio.h>
23  #include <stdlib.h>
24  #include <confuse.h>
25
26  #include <unistd.h>
27  #include <string.h>
28  #include <stdbool.h>
29
30  #include <stdio.h>
31  #include <stdlib.h>
32  #include <unistd.h>
33  #include <dirent.h>
34
35  #include <string.h>
36  #include <sys/time.h>
37  #include <time.h>
```

```
38 #include <sys/param.h>
39 #include <pwd.h>
40 #include <sys/stat.h>
41
42
43 typedef struct utlConteudoConfiguracao_
44 {
45     char nome[UTL_TAM_MAX_VAL_CONF];
46     char valor[UTL_TAM_MAX_VAL_CONF];
47     struct utlConteudoConfiguracao_ *proximo;
48
49 } utlConteudoConfiguracao;
50
51 typedef struct utlListaCOnfiguracao_
52 {
53     char valor[UTL_TAM_MAX_P];
54     struct utlListaCOnfiguracao_ *proximo;
55
56 } utlListaConfiguracao;
57
58 #ifdef UTLCMPAGENTE
59 static float utlObtemTempoDecorrido(void);
60 static unsigned long utlTotalMemKB(void);
61 #endif
62 void utlLeTemperatura(int *temperatura, char *procTemp);
63 void utlUsuarioDoProcesso(int pid, char *nome);
64 void utlDataHoraAtual(char *dataHora);
65 void utlTempoEmSegundos(int *segundos);
66 bool utlExecutarColeta(int tempoEmSegundos);
67 void utlDebug(char *msg, int tipoDeErro, ...);
68 void utlLeConfiguracao(char *nomeDoArquivo, cfg_opt_t opts[]);
69 void utlLeListaConfiguracao(char *nomeDoArquivo, cfg_opt_t opts[], char
   *nomeDaLista, utlListaConfiguracao *utlLC);
70 void utlRemoveListaConfiguracao(utlListaConfiguracao *utlLC);
71 void utlCortaTexto(char *parte, char *texto, char *separador, int posicao);
72
73 #include "util.c"
74
75 #endif
```

- *util.c*: armazena funções utilitárias para as aplicações agented e coletord.

```
1  #ifdef UTLCMPAGENTE
2  static float utlObtemTempoDecorrido(void)
3  {
4     struct timeval t;
5     static struct timeval oldtime;
6     struct timezone timez;
7     float elapsed_time;
8
9     gettimeofday(&t, &timez);
10    elapsed_time = (t.tv_sec - oldtime.tv_sec)
11       + (float) (t.tv_usec - oldtime.tv_usec) / 1000000.0;
12    oldtime.tv_sec  = t.tv_sec;
13    oldtime.tv_usec = t.tv_usec;
14
15    return elapsed_time;
16 }
```

```
17
18 static unsigned long utlTotalMemKB(void)
19 {
20    int len;
21    FILE *meminfo;
22    char buffer[2048], *p;
23    unsigned long memtotal;
24
25    meminfo = fopen("/proc/meminfo", "r");
26    if(!meminfo)
27       return 0;
28
29    len = fread(buffer, sizeof(char), sizeof(buffer)-1, meminfo);
30    buffer[len] = '\0';
31    fclose(meminfo);
32
33    p = (char*) strstr( buffer, "MemTotal:" );
34    if (!p)
35       return 0;
36
37    sscanf(p, "MemTotal: %lu ", &memtotal);
38    return memtotal;
39 }
40 #endif
41
42 void utlLeTemperatura(int *temperatura, char *procTemp)
43 {
44     FILE* status;
45
46     status = fopen(procTemp, "r");
47     if (status)
48     {
49         fscanf(status, "temperature:%d", temperatura);
50     }
51     fclose(status);
52 }
53
54 void utlUsuarioDoProcesso(int pid, char *nome)
55 {
56     struct stat sb;
57     struct passwd* dadosUsuario;
58     int rc, uid;
59
60     char buffer[32];
61     sprintf(buffer, "/%s/%d", PROC_DIR, pid);
62     rc = stat(buffer, &sb);
63     uid = sb.st_uid;
64
65     dadosUsuario = getpwuid(uid);
66     if (dadosUsuario)
67     {
68         sprintf(nome, "%s", dadosUsuario->pw_name);
69     }
70     else
71     {
72         sprintf(nome, "%d", uid);
73     }
74 }
75
76 void utlDataHoraAtual(char *dataHora)
77 {
78     struct timeval tv;
79     struct tm* ptm;
80     char time[40];
81
82
83     gettimeofday(&tv, NULL);
```

```
84      ptm = localtime(&tv.tv_sec);
85      strftime(time, sizeof(time), "%Y%m%d %H%M%S", ptm);
86      sprintf(dataHora, "%s", time);
87  }
88
89  void utlTempoEmSegundos(int *segundos)
90  {
91      struct timeval tv;
92      gettimeofday(&tv, NULL);
93
94      *segundos = tv.tv_sec;
95  }
96
97  bool utlExecutarColeta(int tempoEmSegundos)
98  {
99      int tempoAtualEmSegundos;
100     utlTempoEmSegundos(&tempoAtualEmSegundos);
101
102     if (tempoEmSegundos <= tempoAtualEmSegundos)
103     {
104         return true;
105     }
106
107     return false;
108 }
109
110 void utlDebug(char *msg, int tipoDeErro, ...)
111 {
112     if (UTL_EXIBE_NOTICIA == 0 && tipoDeErro == UTL_NOTICIA)
113     {
114         return;
115     }
116
117     switch (tipoDeErro)
118     {
119         case UTL_NOTICIA:    printf("NOTÍCIA: "); break;
120         case UTL_CUIDADO:    printf("CUIDADO: "); break;
121         case UTL_ERRO_FATAL: printf("ERRO FATAL: "); break;
122     }
123
124     va_list argv;
125     va_start(argv, tipoDeErro);
126     while (*msg)
127     {
128         char caract = *msg;
129
130         if (caract != '%')
131         {
132             printf("%c", caract);
133             msg++;
134             continue;
135         }
136         msg++;
137         switch (*msg)
138         {
139             case 'c': printf("%c", va_arg(argv, int)); break;
140             case 's': printf("%s", va_arg(argv, char*)); break;
141             case 'd': printf("%d", va_arg(argv, int)); break;
142             default: printf("%c", caract);
143         }
144         msg++;
145     }
146     printf("\n");
147     va_end(argv);
148     if (tipoDeErro == UTL_ERRO_FATAL)
149     {
150         exit(1);
```

```
151     }
152 }
153
154 void utlLeConfiguracao(char *nomeDoArquivo, cfg_opt_t opts[])
155 {
156     utlDebug("Lendo arquivo de configuração '%s'", UTL_NOTICIA, nomeDoArquivo);
157
158     int ret;
159
160     cfg_t *cfg;
161     cfg = cfg_init (opts, 0);
162     ret = cfg_parse (cfg, nomeDoArquivo);
163
164     if(ret == CFG_FILE_ERROR)
165     {
166         utlDebug("Não foi possível ler o arquivo '%s'.", UTL_ERRO_FATAL,
    nomeDoArquivo);
167     }
168     else if(ret == CFG_PARSE_ERROR)
169     {
170         utlDebug("Erro de sintaxe no arquivo '%s'.", UTL_ERRO_FATAL,
    nomeDoArquivo);
171     }
172     cfg_free(cfg);
173 }
174
175 void utlLeListaConfiguracao(char *nomeDoArquivo, cfg_opt_t opts[], char
    *nomeDaLista, utlListaConfiguracao *utlLC)
176 {
177     utlDebug("Lendo arquivo de configuração '%s'", UTL_NOTICIA, nomeDoArquivo);
178
179     int ret, i;
180
181     utlListaConfiguracao *novo;
182     cfg_t *cfg;
183     cfg = cfg_init (opts, 0);
184     ret = cfg_parse (cfg, nomeDoArquivo);
185
186     if(ret == CFG_FILE_ERROR)
187     {
188         utlDebug("Não foi possível ler o arquivo '%s'.", UTL_ERRO_FATAL,
    nomeDoArquivo);
189     }
190     else if(ret == CFG_PARSE_ERROR)
191     {
192         utlDebug("Erro de sintaxe no arquivo '%s'.", UTL_ERRO_FATAL,
    nomeDoArquivo);
193     }
194
195     for(i = 0; i < cfg_size(cfg, nomeDaLista); i++)
196     {
197         if ( i > 0 )
198         {
199             novo = (utlListaConfiguracao*) malloc(sizeof(utlListaConfiguracao));
200             utlLC->proximo = novo;
201             utlLC = utlLC->proximo;
202         }
203         strcpy(utlLC->valor, cfg_getnstr(cfg, nomeDaLista, i));
204         utlLC->proximo = (utlListaConfiguracao*) NULL;
205     }
206     printf("\n");
207
208     cfg_free(cfg);
209 }
210
211 void utlRemoveListaConfiguracao(utlListaConfiguracao *utlLC)
212 {
```

```
213     utlListaConfiguracao *auxiliar;
214     utlDebug("Remove lista de configuração", UTL_NOTICIA);
215     while (utlLC != (utlListaConfiguracao*) NULL)
216     {
217         auxiliar = utlLC;
218         utlLC = utlLC->proximo;
219         free(auxiliar);
220     }
221 }
222
223 void utlCortaTexto(char *parte, char *texto, char *separador, int posicao)
224 {
225     int parteDoTexto = 0;
226     int x = 0;
227     char *inicioSeparador = separador;
228
229     while (*texto)
230     {
231         if (*texto == *separador)
232         {
233             separador++;
234             if (!*separador)
235             {
236                 if (parteDoTexto == posicao)
237                 {
238                     break;
239                 }
240                 parteDoTexto++;
241                 x = 0;
242                 separador = inicioSeparador;
243             }
244         }
245         else
246         {
247             parte[x] = *texto;
248             x++;
249         }
250         texto++;
251     }
252     if (parteDoTexto < posicao)
253     {
254         x = 0;
255     }
256     parte[x] = '\0';
257 }
```

APÊNDICE E SQLs para criação do banco de dados

Tabela *processos*:

```
1  CREATE TABLE processos ( id SERIAL,
2                           nomeDoHost varchar(50),
3                           pid integer,
4                           dataHoraInicial timestamp,
5                           dataHoraFinal timestamp,
6                           nome varchar(50),
7                           usuario varchar(50),
8                           utilizacaoDaCPU float,
9                           utilizacaoDaMemoria float );
```

Tabela *temperaturas*:

```
1  CREATE TABLE temperaturas ( id SERIAL,
2                              nomeDoHost varchar(50),
3                              dataHoradaColeta timestamp,
4                              temperatura integer);
```

Tabela *particoes*:

```
1  CREATE TABLE particoes ( id SERIAL,
2                           nomeDoHost varchar(50),
3                           dataHoradaColeta timestamp,
4                           nome varchar(50),
5                           pontoDeMontagem varchar(50),
6                           totalEmKb float,
7                           usadoEmKb float);
```

APÊNDICE F Makefile

```
1  CC=gcc
2
3  TARGETS=coletord agented
4
5  CFLAGS=-I. `net-snmp-config --cflags` -I`pg_config --includedir` `confuse-config
   --cppflags` -I../include -Wall
6  BUILDLIBS=`net-snmp-config --libs` `net-snmp-config --agent-libs` -L`pg_config --
   libdir` `confuse-config --libs` -lpq
7  OBJS=agented.o tccProcessTable.o tccTemperatureTable.o tccPartitionTable.o
8
9  # shared library flags (assumes gcc)
10 DLFLAGS=-fPIC -shared
11
12 #Por padrão compila todos os programas
13 defaul: all
14
15 #Exclui arquivos compilados e temporários
16 clean:
17         rm -f *.o *~ $(TARGETS) *.so
18
19 #Compila os programas definidos na variável TARGETS
20 all: $(TARGETS)
21
22 #Compila o programa coletor
23 coletord: coletord.o
24         $(CC) coletord.o -o coletord $(BUILDLIBS)
25 coletord.o: coletord.c
26         $(CC) coletord.c -c $(CFLAGS)
27
28 #Compila o programa agente
29 agented: $(OBJS)
30         $(CC) -o agented $(OBJS)  $(BUILDLIBS)
31
32 agented.o: agented.c
33         $(CC) $(CFLAGS) $(DLFLAGS) -c -o agented.o agented.c
34         $(CC) $(CFLAGS) $(DLFLAGS) -o agented.so agented.o
```