PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL - PUCRS

Escola Politécnica

Henrique Corrêa

Joice Marek

Gerência de Redes de Computadores

Trabalho Final

AGENTE SNMP ESTENDIDO

Porto Alegre, 22 de outubro de 2018

INTRODUÇÃO

O SNMP (*Simple Network Management Protocol*) ou Protocolo Simples de Gerenciamento de Redes, como o próprio nome sugere, foi criado para facilitar o gerenciamento de redes, definindo como o gerente se comunica com um agente. Atualmente é o protocolo padrão utilizado para gerenciamento e monitoração de redes.

Este trabalho visa estudar o processo de gerenciamento para monitoração dos elementos de uma rede de computadores. O estudo envolve o desenvolvimento de um agente SNMP estendido com uma MIB definida pelo grupo.

ESPECIFICAÇÃO DA MIB

A MIB implementada possui sete objetos que fazem parte de três áreas funcionais, sendo elas: desempenho, falhas e configurações. A Figura 1 apresenta a estrutura da MIB estendida. Como pode-se observar os objetos se encontram no nodo **Experimental**, onde o nodo **GerRedes** representa a “corporação” e o nodo **process** corresponde ao grupo de objetos. Por fim, os nodos folhas são os objetos implementados. Sendo assim, para acessá-los deve-se referenciar o OID (ID do objeto) que é 1.3.6.1.3.17.1.<número do objeto>.

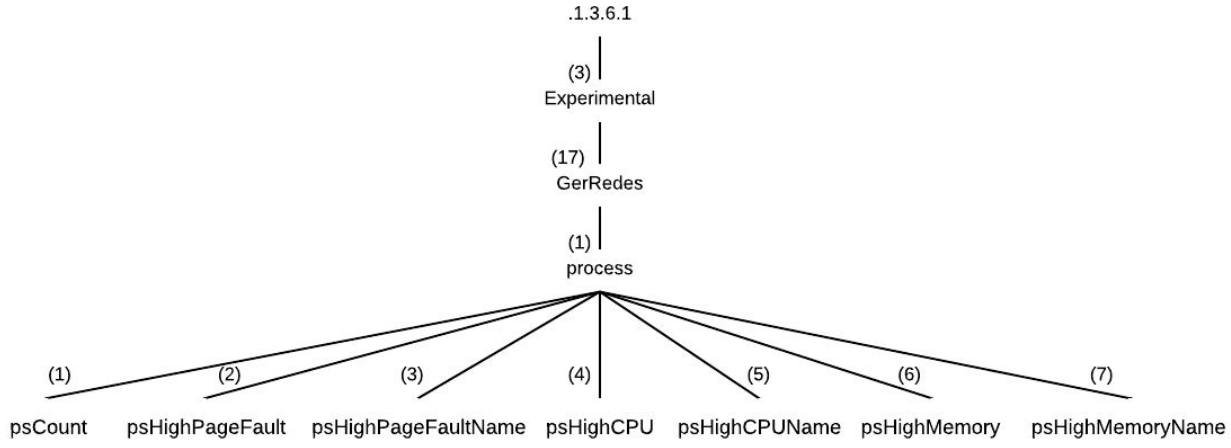


Figura 1: Estrutura da MIB estendida

A seguir é apresentada a definição de cada objeto:

- **Objeto 1)**
  - **Nome:** psCount
  - **Tipo de retorno:** Counter
  - **Tipo de acesso:** Read-only
  - **Área funcional:** Desempenho

    - **Descrição:** Número de processos em execução no sistema
- **Objeto 2)**
    - **Nome:** psHighPageFault
    - **Tipo de retorno:** Counter
    - **Tipo de acesso:** Read-only
    - **Área funcional:** Falhas
    - **Descrição:** Maior valor de page fault dos processos em execução
- **Objeto 3)**
    - **Nome:** psHighPageFaultName
    - **Tipo de retorno:** String
    - **Tipo de acesso:** Read-only
    - **Área funcional:** Falhas
    - **Descrição:** Nome do processo com maior valor de page fault
- **Objeto 4)**
    - **Nome:** psHighCPU
    - **Tipo de retorno:** String
    - **Tipo de acesso:** Read-write
    - **Área funcional:** Desempenho/Configuração
    - **Descrição:** Retorna o maior valor de uso de CPU em porcentagem dos processos em execução. Configura o uso da CPU para a aplicação por 2 minutos.
- **Objeto 5)**
    - **Nome:** psHighCPUName
    - **Tipo de retorno:** String
    - **Tipo de acesso:** Read-only
    - **Área funcional:** Desempenho
    - **Descrição:** Nome do processo com maior valor de uso de CPU dos processos em execução.
- **Objeto 6)**
    - **Nome:** psHighMemory
    - **Tipo de retorno:** Counter
    - **Tipo de acesso:** Read-only
    - **Área funcional:** Desempenho
    - **Descrição:** Maior valor de uso de memória dos processos em execução.

- **Objeto 7)**
    - **Nome:** psHighMemoryName
    - **Tipo de retorno:** String
    - **Tipo de acesso:** Read-only
    - **Área funcional:** Desempenho
    - **Descrição:** Nome do processo com maior valor de uso de memória.

CONFIGURAÇÃO DO AMBIENTE

Para este trabalho, foi instalado as aplicações e os agentes snmp para um sistema operacional linux com o gerenciador de pacotes **apt**, nativo em distros com base o Debian. Também foi utilizado os mesmos meios para instalar e configurar a MIBv2 no ambiente.

Configuração MIB

O agente possui um diretório padrão para procurar novas MIBs localizado em ***$HOME/.snmp/mibs***. Nesta pasta, criou-se o arquivo **GerRedes-MIB.txt** contendo a especificação da MIB descrita no capítulo anterior. Para o agente carregar a MIB, também foi necessário modificar o arquivo */etc/snmp/***snmp.conf** com a seguinte instrução:

```
mibs +GerRedes-MIB
```

A instrução manda o agente utilizar todas as MIBs que ele já conhece e adicionar mais uma mib, que seria a **GerRedes-MIB**. Vale salientar que a instrução não se refere ao nome do arquivo **GerRedes-MIB.txt**, mas sim a definição presente dentro dele:

```
GerRedes-MIB DEFINITIONS ::= BEGIN
```

Configuração agente

Para permitir a extensão do agente, foi adicionado uma nova instrução no arquivo de configuração do agente */etc/snmp/***snmpd.conf**:

```
pass .1.3.6.1.3.17 /bin/sh $HOME/.snmp/scripts/passGerRedes
```

A instrução **pass** faz com que, para requisições com o OID abaixo de .1.3.6.1.3.17, será invocado o programa /bin/sh com o endereço do script que implementa os objetos da MIB em *$HOME/.snmp/scripts/***passGerRedes** (OBS: trocar a variável $HOME pelo caminho absoluto do script).

EXTENSÃO DO AGENTE

O agente foi estendido com um script bash e responde as operações GET, GET-NEXT e SET. Os objetos foram implementados utilizando o comando **ps**. Para retornar os valores utiliza-se o comando **echo**.

Quando é feita uma requisição de GET, é verificado se a instância do objeto é válida, para assim, retornar um objeto existente na MIB estendida. Para o caso de um GET-NEXT, verifica-se a próxima instância válida para se definir como o objeto de retorno. Com as validações bem sucedidas, é retornado o ID e o tipo do objeto requerido (ou do próximo objeto para GET-NEXT) e então é chamada a função que irá retornar o valor do objeto.

No caso de uma requisição SET, o agente escreve o valor enviado pelo gerente em um arquivo temporário para ser lido e executado por um outro script auxiliar que deve ser inicializado pelo host que está recebendo a requisição. Justifica-se este procedimento pelo fato que o agente não tem o privilégio necessário para implementar diretamente o objeto read-write *psHighCPU*, além do risco implícito por não exigir nenhum controle do usuário que está realizando a requisição.

Funcionamento script de extensão

Inicialmente, o script define OID de todos os objetos a ser implementados, além de inicializar variáveis auxiliares:

```
##########################
#MIB Object ID Definition
##########################

PLACE=".1.3.6.1.3.17"  # experimental.GerRedes
GERREDES_OID=".1.3.6.1.3.17"  # experimental.GerRedes
PSINFO_OID="$GERREDES_OID.1"  # experimental.GerRedes.process
PSCOUNT_OID="$PSINFO_OID.1"  # experimental.GerRedes.process.psCount
PSHIGHPAGEFAULT_OID="$PSINFO_OID.2"  #
experimental.GerRedes.process.psHighPageFault
PSHIGHPAGEFAULTNAME_OID="$PSINFO_OID.3"  #
experimental.GerRedes.process.psHighPageFaultName
PSHIGHCPU_OID="$PSINFO_OID.4"  # experimental.GerRedes.process.psHighCPU
PSHIGHCPUNAME_OID="$PSINFO_OID.5"  #
experimental.GerRedes.process.psHighCPUName
PSHIGHMEMORY_OID="$PSINFO_OID.6"  #
experimental.GerRedes.process.psHighMemory
PSHIGHMEMORYNAME_OID="$PSINFO_OID.7"  #
experimental.GerRedes.process.psHighMemoryName
```

```
########################
# Auxiliary variables
########################

REQ="$2"                          # Requested OID
RESULT=false                      # Boolean flag used to inform if arguments
are valid
FILE_SNMP_PASS_SET="/tmp/passsnmpset.log" #Define the expected file to be
created by SNMP agent
```

Quando o agente executa o script, são utilizados os argumentos:

- $1 - Tipo de requisição (GET = “-g”; GETNEXT = “-n”; SET = “-s”)
- $2 - OID solicitado (Ex: .1.3.6.1.3.17.1.0)
- $3 (apenas para SET) - Tipo de dado a ser escrito (Ex: s = String, i = Integer)
- $4 (apenas para SET) - Dado a ser escrito (Ex:”Hello world”)

Após sua inicialização, o programa verifica qual é o tipo de requisição e faz a verificação apropriada dos argumentos recebidos. Caso a verificação falhe (Ex: OID não existe), é atribuído o valor **falso** para a variável auxiliar **RESULT**.

```
################################
#  GETNEXT requests - determine next valid instance
################################

if [ "$1" = "-n" ]; then

  case $REQ in
      $PLACE|\
      $PLACE.0|\
      $PLACE.0.*|\
      $PLACE.1|\
      $PLACE.1.0|\
      $PLACE.1.0.*|\
      $PLACE.1.1)  RET=$PSCOUNT_OID.0; RESULT=true ;;

      $PLACE.1.1.*|\
      $PLACE.1.2) RET=$PSHIGHPAGEFAULT_OID.0; RESULT=true ;;

      $PLACE.1.2.*|\
      $PLACE.1.3) RET=$PSHIGHPAGEFAULTNAME_OID.0; RESULT=true ;;

      $PLACE.1.3.*|\
      $PLACE.1.4) RET=$PSHIGHCPU_OID.0; RESULT=true ;;

      $PLACE.1.4.*|\
```

```
      $PLACE.1.5) RET=$PSHIGHCPUNAME_OID.0; RESULT=true ;;

      $PLACE.1.5.*|\
      $PLACE.1.6) RET=$PSHIGHMEMORY_OID.0; RESULT=true ;;

      $PLACE.1.6.*|\
      $PLACE.1.7) RET=$PSHIGHMEMORYNAME_OID.0; RESULT=true ;;
  esac

fi

########################################
## SET requests
########################################
if [ "$1" = "-s" ]; then

  case "$REQ" in
      $PSHIGHCPU_OID.0) echo "$4" >> "$FILE_SNMP_PASS_SET";; #Save set
value in a file
      *)  exit 0;;
  esac

  exit 0
fi

########################################
#  GET requests - check for valid instance
########################################
if [ "$1" = "-g" ]; then
  case "$REQ" in
      $PSCOUNT_OID.0|\
      $PSHIGHPAGEFAULT_OID.0|\
      $PSHIGHPAGEFAULTNAME_OID.0|\
      $PSHIGHCPU_OID.0|\
      $PSHIGHCPUNAME_OID.0|\
      $PSHIGHMEMORY_OID.0|\
      $PSHIGHMEMORYNAME_OID.0)   RET=$REQ; RESULT=true ;;   #Requested
Object ID is known and valid
  esac
fi
```

Para as solicitações de leitura GET e GETNEXT, é retornado o OID do objeto solicitado, o tipo de dado a ser retornado (Ex: Counter, String) e o dado (Ex: “Hello World”).

```
########################################
# Request return
########################################

#If Object ID parse was successfull, process requested operation
```

```
if "$RESULT" ; then
  echo "$RET";     #Return Object ID

  #Return data type and call function to return data
  case "$RET" in
      $PSCOUNT_OID.0)    echo "counter"; function_pscount;;
    $PSHIGHPAGEFAULT_OID.0)    echo "counter";  function_pshighpagefault;;
      $PSHIGHPAGEFAULTNAME_OID.0)    echo "string";
function_pshighpagefaultname;;
      $PSHIGHCPU_OID.0) echo "string"; function_pshighcpu;;
      $PSHIGHCPUNAME_OID.0) echo "string"; function_pshighcpuname;;
      $PSHIGHMEMORY_OID.0) echo "counter"; function_pshighmemory;;
      $PSHIGHMEMORYNAME_OID.0) echo "string"; function_pshighmemoryname;;
      *)  exit 0;;# Default value (should not happen)
  esac
fi
```

Para cada tipo de objeto, uma função é chamada, sendo responsável por retornar o dado correspondente a solicitação.

```
############################
#Function declaration
############################

function_pscount(){
  echo $( ps -A --no-headers | wc -l);
}

function_pshighpagefault(){
  echo $( ps -eo maj_flt --sort=maj_flt | tail -n 1 | tr -d [:space:] );
}

function_pshighpagefaultname(){
  echo $( ps -eo comm --sort=maj_flt | tail -n 1 | tr -d [:space:] );
}

function_pshighcpu(){
  echo $(ps -eo pcpu --sort=-pcpu --no-headers | head -n 1);
}

function_pshighcpuname(){
  echo $(ps -eo comm --sort=-pcpu --no-headers | head -n 1);
}

function_pshighmemory(){
  echo $(ps -eo pmem --sort=-pmem --no-headers | head -n 1);
}

function_pshighmemoryname(){
```

```
  echo $(ps -eo comm --sort=-pmem --no-headers | head -n 1);
}
```

Funcionamento script auxiliar

Conforme mencionado, o script de extensão escreve os dados de uma requisição SET em um arquivo temporário */tmp/***passsnmpset.log**. O conteúdo deste arquivo é lido pelo script auxiliar $HOME/.snmp/scripts/**setHandler.sh**. É responsabilidade do usuário inicializar este script para atender as requisições, pois esta é a medida de segurança que impede o agente de atuar sobre o sistema por conta própria e de forma indevida.

Em sua inicialização, o programa tenta ler o arquivo do agente e, caso bem sucedido, grava quantas linhas de informação já foram inseridas para não processar instruções anteriores a sua execução.

```
#################
#program setup
#################

#Define the expected file to be created by SNMP agent
FILE_SNMP_PASS_SET="/tmp/passsnmpset.log"

#Check the number of records present in snmp logfile
if [ -e "$FILE_SNMP_PASS_SET" ]; then
       PREVIOUS_LINE_COUNT=$(cat "$FILE_SNMP_PASS_SET" | wc -l)
else
    #File not found
       PREVIOUS_LINE_COUNT=0
fi

#Sync actual index counter with read value
ACTUAL_LINE_COUNT=$PREVIOUS_LINE_COUNT
```

Concluído, entra-se em loop infinito observando a inserção de novos dados no arquivo:

```
################
#program start
################

while true; do

       #Check if file exists or has been removed
       if [ -e "$FILE_SNMP_PASS_SET" ]; then

       #Update line counter
```

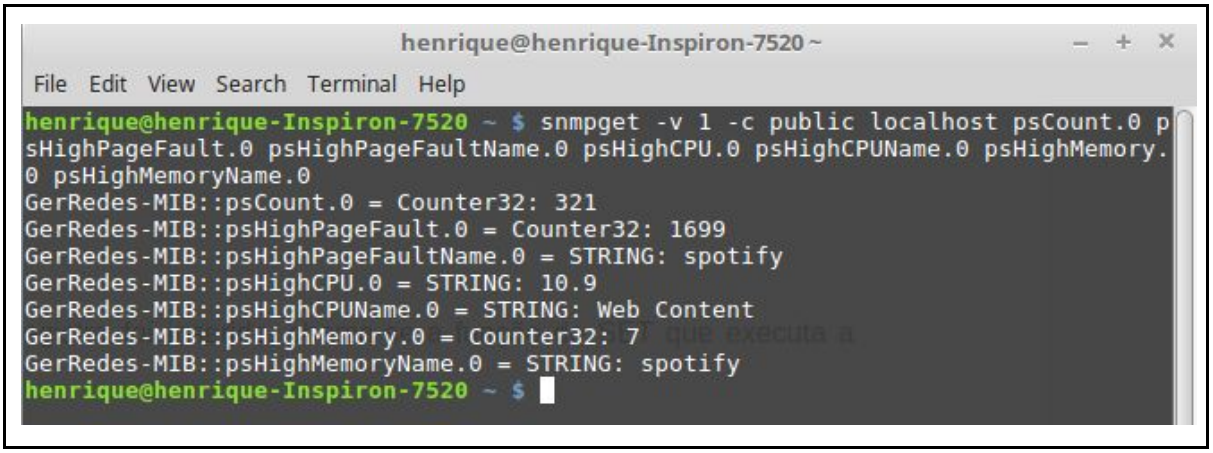
```
	ACTUAL_LINE_COUNT=$(cat "$FILE_SNMP_PASS_SET" | wc -l)

	#If file countains new records, process each one of them
	while [ $PREVIOUS_LINE_COUNT -lt $ACTUAL_LINE_COUNT ]; do
		PREVIOUS_LINE_COUNT=$(expr $PREVIOUS_LINE_COUNT + 1)
#Increment counter

		#Execute implemented functions
		function_set_pshighcpu

	done
	else
	#File not found: restart counters
	PREVIOUS_LINE_COUNT=0
	ACTUAL_LINE_COUNT=0
	fi

	sleep 1 #Wait a while to execute this routine again
done
```

Quando um novo registro foi inserido, chama-se a função de SET que executa a operação no sistema:

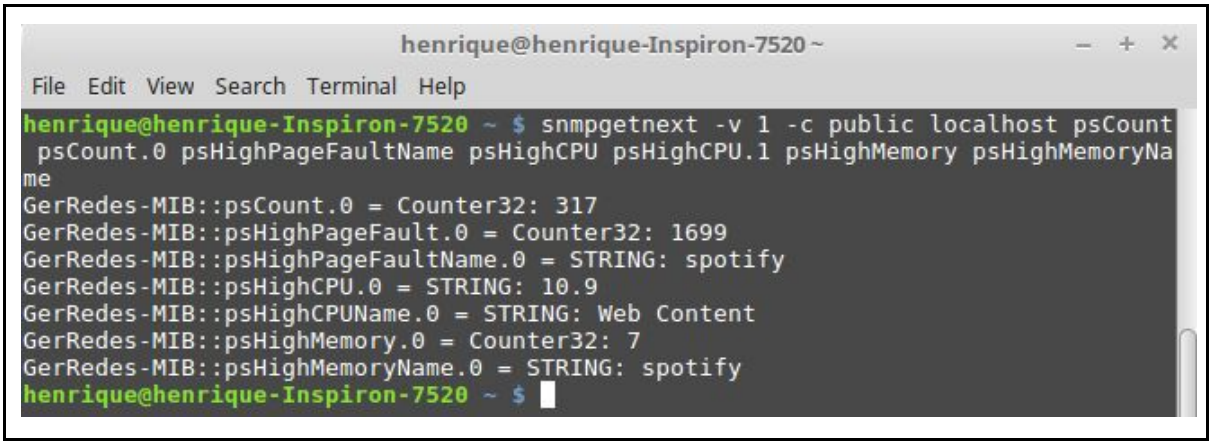
```
############################
#Function declaration
############################

function_set_pshighcpu(){

  PROC_PID=$( ps -eo pid --sort=-pcpu --no-headers | head -n 1 ); #Get
PID of the process with highest CPU usage
  CPU_LIMIT_VALUE=$(sed -n "$PREVIOUS_LINE_COUNT"p "$FILE_SNMP_PASS_SET")
#Get the cpulimit value set by the user through the SNMP agent log
  echo timeout 20s cpulimit -p $PROC_PID -l $CPU_LIMIT_VALUE #print to the
user the command to be executed
  timeout 20s cpulimit -p $PROC_PID -l $CPU_LIMIT_VALUE &  #Limit the
process with the CPU usage set by the user through the SNMP agent log for
20 seconds
}
```

DEMONSTRAÇÃO E RESULTADOS

Resultados obtidos para solicitações GET dos objetos da MIB GerRedes:

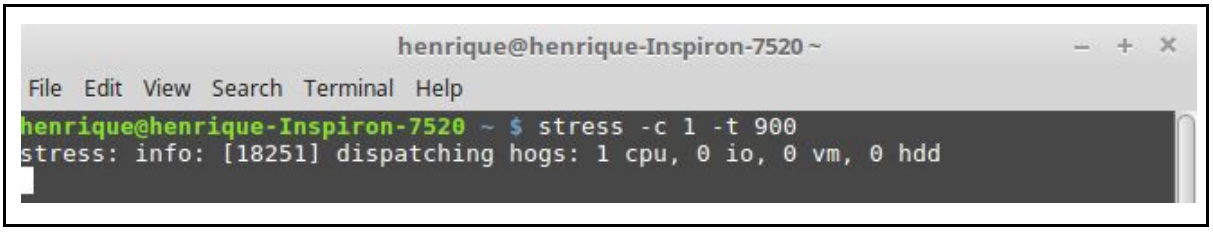


Resultados obtidos para solicitações GETNEXT dos objetos da MIB GerRedes:

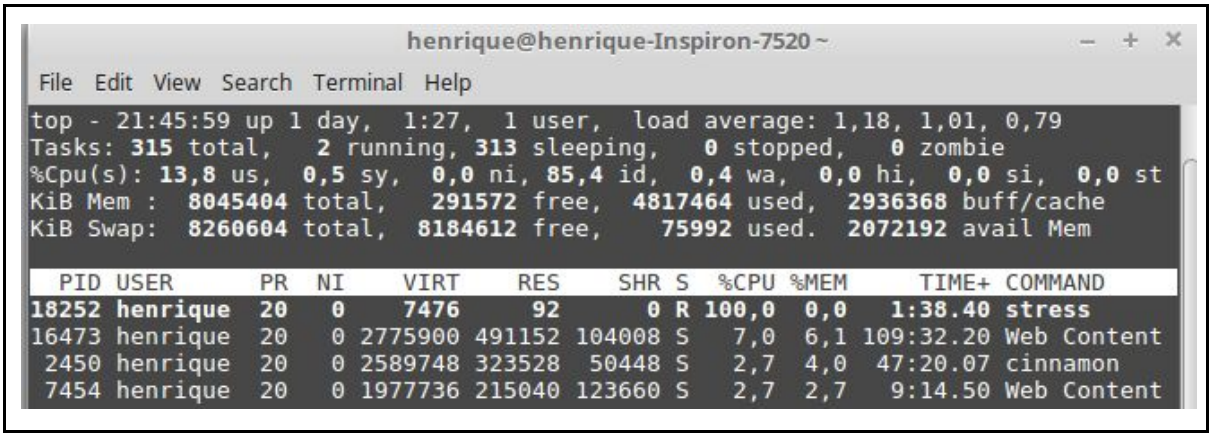


Para realizar o teste SET do objeto psHighCPU, primeiramente foi aberto um terminal que inicializa o script auxiliar **setHandler.sh**:

Em um novo terminal, inicializou-se o programa *stress* para provocar um alto uso da CPU por 900 segundos:

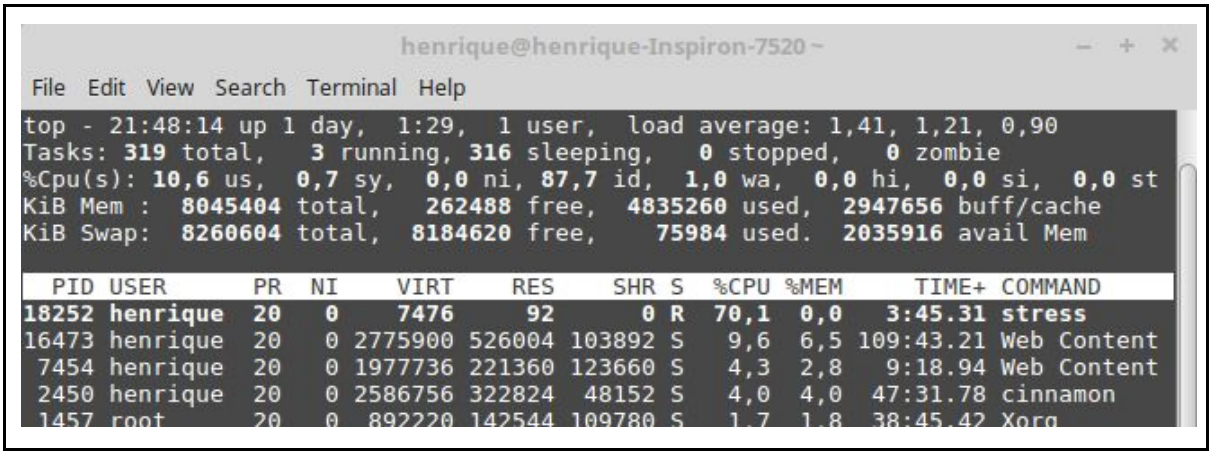


Na sequência, executa-se o programa *top* para realizar o monitoramento de uso da CPU. Como esperado, o processo *stress* está consumindo 100% da CPU.



Por fim, é realizada a requisição SNMP para modificar o uso da CPU para 70%:

Observa-se no terminal que está executando *top* que o uso da CPU do processo *stress* foi reduzido para um valor próximo de 70%.



CONCLUSÃO

Este trabalho apresentou uma de diversas formas de extensão de agentes SNMP. A maior adversidade encontrada no desenvolvimento do trabalho foi configurar o ambiente para que o agente utilize o script que implementa os objetos da nova MIB. Depois que esta barreira foi vencida, foi simples a implementação dos objetos da MIB. Durante as pesquisas,

encontramos outras alternativas para extensão do agente como scripts em python, perl, c ou até mesmo a criação de sub-agentes. Sendo assim, concluímos que não existe a melhor forma para se estender um agente, isso vai da familiaridade do programador com a linguagem escolhida e a interface que será utilizada para atender às requisições do agente.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] Douglas E. COMER. Redes de Computadores e Internet. 6. ed. Porto Alegre: Bookman, 2016.

[2] OpServices. *Entenda o que é o protocolo SNMP e sua importância no monitoramento*. 2017. Disponível em: <https://www.opservices.com.br/snmp/>.

[3] Slides da aula. Gerência-MIB-SMI.

[4] Net-SNMP. *Tutorials*. 2013. Disponível em: <http://net-snmp.sourceforge.net/wiki/index.php/Tutorials>

[5] Net-SNMP. *TUT:snmptranslate*. 2011. Disponível em: <http://net-snmp.sourceforge.net/wiki/index.php/TUT:snmptranslate>

[6] Net-SNMP. *TUT:Using and loading MIBS*. 2016. Disponível em: <http://net-snmp.sourceforge.net/wiki/index.php/TUT:Using_and_loading_MIBS>