Universidade do Minho

Mestrado Integrado em Engenharia Informática

Gestão e Virtualização de Redes

Gestão de Redes

Ficha de Trabalho Prático Nº2

Ferramenta de Monitorização

Joana Isabel Afonso Gomes **a84912**

28 de Fevereiro de 2021

Conteúdo

Lista de Figuras

1 INTRODUÇÃO

No âmbito da Unidade Curricular de Gestão de Redes foi-me proposto este segundo trabalho prático, que consiste na criação de uma ferramenta **SNMP** (Simple Network Management Protocol) que permita a monitorização e análise da utilização dos recursos do sistema local pelos processos ativos num *host*.

No presente relatório apresentarei e justificarei as opções tomadas na sua implementação, mostrando as várias vertentes do programa e a forma como decorreu o seu desenvolvimento.

2 ESTRUTURA DA FERRAMENTA

Com vista ao objetivo apresentado, neste projeto foram desenvolvidos dois módulos.

O primeiro módulo passa pela implementação do mecanismo de monitorização, que gera como *output* ficheiros *log* com dados de gestão sobre a utilização de recursos de **CPU** e Memória **RAM**, recorrendo a um ficheiro como forma de configuração. Para este módulo utilizei a linguagem ***Java*** e recorri à API ***SNMP4J*** para obter as primitivas do SNMP.

O segundo módulo tem como objetivo a análise dos dados nos ficheiros *logs* gerados no Módulo 1, recorrendo a uma interface gráfica interativa (desenvolvida em ***JavaFX***) que exibe diversas opções para observação de dados relevantes no que toca à utilização de recursos dos processos ativos nos *hosts*, como gráficos, tabelas e métodos de pesquisa por filtragem.

De seguida apresentarei cada módulo e os seus métodos e funcionalidade mais relevantes, assim como as intruções de utilização e os resultados obtidos.

3 IMPLEMENTAÇÃO DO MECANISMO DE MONITORIZAÇÃO

Nesta secção indicarei com detalhe o processo de desenvolvimento do Módulo 1, especificando os passos necessários para a obtenção do *output* dos dados de gestão.

Neste módulo existem duas classes: `User` e `Client`. O objetivo é obter endereços e portas dos *hosts* que são fornecidos pelo utilizador segundo o *input* no ficheiro `config.json` para poder ativar o mecanismo de de monitorização SNMP para os mesmos.

Juntamente com o endereço e a porta de um ou mais *hosts*, no ficheiro de configuração *json* é também incluído o intervalo de monitorização, optando eu assim por que o intervalo de monitorização da minha ferramenta fosse configurável pelo utilizador.

```
{
  "Hosts": [
    {
      "Address": "127.0.0.1",
      "Port": "161"
    },
    {
      "Address": "127.0.0.2",
      "Port": "161"
    }
  ],
  "Interval": "40"
}
```

Listing 1: Exemplo de *input* no ficheiro `config.json`

Para testar a ferramenta na sua vertente de monotorização para um **ou mais** *hosts* recorri aos dois endereços que estão no ficheiro *json* exemplo apresentado em cima, após ter chegado à conclusão que o outro endereço de *localhost* também funcionava e evitando assim, por exemplo, a utilização de uma *Virtual Machine*.

Na classe `Client` existem dois importantes métodos: `doWalk` e `snmpWalk`. O `doWalk` tem como objetivo criar um *SNMP walk* das interfaces de um dispositivo. O código deste método foi inspirado em [3] e recebe como parâmetros um *Table OID* e um objeto *CommunityTarget*. Esta *CommunityTarget* será criada aquando da criação de uma instância da classe `Client`, através do método `configTarget`, e contém a *community string*, o endereço *target* IP, a porta e outros dados.

```
public Client(String address,String port) throws IOException {
    this.address=address;
    this.port=Integer.parseInt(port);

    this.target = new CommunityTarget();
    configTarget();

    this.transport = new DefaultUdpTransportMapping();
    this.transport.listen();
}
```

Listing 1: Construtor da classe `Client`

```
public void configTarget(){
    this.target.setCommunity(new OctetString(community));
    this.target.setVersion(snmpVersion);
    this.target.setAddress(new UdpAddress(this.address + "/" + port));
    (...)
}
```

Listing 2: Método `configTarget()`

Depois da implementação do `doWalk`, prossegui à criação do método *snmpWalk*, que faz um *SNMP walk* (recorrendo ao método `doWalk`) nos OIDs escolhidos, criando *Maps* e returnando a String com o resultado que serão as linhas imprimidas no ficheiro *log*, com informação relativa dos segundos a que foi feita a monitorização referente aquela linha do *log file* (com referência no momento em que começou a monitorização) e à utilização da Memória RAM e do CPU.

Para obter estes dados, utilizei os seguintes OIDs de objetos da *Host-Resources-MIB*:

- `hrSWRunName` (.1.3.6.1.2.1.25.4.2.1.2) - indica o nome dos processos ativos;
- `hrSWRunPerfMem` (.1.3.6.1.2.1.25.5.1.1.2.) - indica o total de memória RAM do sistema alocada a cada processo (em KBytes);

- `hrSWRunPerfCPU` (.1.3.6.1.2.1.25.5.1.1.1.) - indica o número (em centisegundos) do total de recursos de CPU consumidos por cada processo.

A classe `User` corresponde à implementação destes métodos nos *hosts* indicados no ficheiro *config.json*. Para tal foi criado o método `inputUser` que lê a informação do ficheiro JSON e cria um *Map* dos *Hosts*.

De seguida implementei o método `monotoring` que usa o `snmpWalk` do *Client* (previamente descrito), criando uma *thread* para cada *host* no ficheiro de configuração e fazendo a monitorização com um intervalo que é, como já havia referido, dado como *input* pelo utilizador.

O resultado é escrito num ficheiro *log* que escolhi ter como nome o endereço do *host* a que corresponde, concatenado com a data e hora a que se iniciou o *monitoring* dos *hosts*.

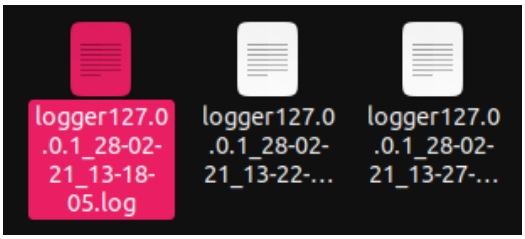

Figura 1: Exemplo de nome de um ficheiro *log* criado pela ferramenta.

A monitorização pára e são criados os *logs* aquando da inserção da palavra "`stop`"no terminal.

4 PROGRAMA DE ANÁLISE DOS DADOS

O Módulo 2 tem o objetivo, como já referido anteriormente, de análise dos *logs* gerados no Módulo 1 e a apresentação de dados relevantes no que toca à utilização do CPU e da memória RAM.

Para a criação de uma **interface** interativa e *user-friendly* optei por recorrer à plataforma ***JavaFX***. Deste módulo fazem então parte as classes ***Controller*** e ***Main*** e os vários ficheiros *.fxml* criados para o desenvolvimento da aplicação gráfica.

De entre os métodos criados, destaco primeiramente o método ***processData*** da classe *Main*, que **guarda e processa a informação do *log file*** que é escolhido pelo utilizador para ser analisado na aplicação.

Na classe *Controller* existem os mais diversos métodos para tratamento de dados e correto *display* da informação na aplicação (claro está, para além de todas as necessidades de implementação para o correto funcionamento da aplicação).

A aplicação inicia-se, como já referido, com o pedido da **escolha** pelo utilizador de um ***log file*** para ser analisado, através de um ***FileChooser*** (no caso da minha ferramente os ficheiros *log* são guardados na *folder parent* da pasta que contém o projeto do Módulo 2, localização que escolhi para a criação dos mesmos no Módulo 1, sendo também a sua *parent folder*).

De seguida, é apresentado um **Menu** com as opções de escolha para o utilizador no que toca às vertentes da ferramenta.

Um primeiro botão é relativo à apresentação em **tabela** dos dados já tratados, em colunas que indicam o nome do processo (devo referir que escolhi deixar sempre o nome do processo concatenado com o seu PID), o número de recursos de CPU consumidos por cada processo e o total de memória RAM do sistema alocada a cada um. No *display* dos dados na aplicação, optei por apresentar os dados do CPU em segundos e da RAM em Bytes (efetuando as necessárias conversões). Para ser possível apresentar corretamente a tabela, desenvolvi os métodos **populateProcTable** e **showTable**, através de entidades

da classe `Entrada`, que criei de forma a compatibilizar a operação detalhada de obter a informação a dar *display* com a inserção nas tabelas JavaFX, inspirando-me na explicação de [2]. Na mesma *window* é ainda possível **filtrar a pesquisa por** um determinado **nome de um processo**.

As outras opções do Menu referem-se ao *display* de **gráficos** que achei relevantes para um administrador de um *host*. Escolhi por isso dar *display* de gráficos que mostram os **5 processos que atingiram os valores maiores de CPU** e os **5 que atingiram os valores maiores de RAM**. Para tal utilizei a funcionalidade ***AreaChart*** do JavaFX e desenvolvi os métodos **graphFiveMoreCPU** e **graphFiveMoreRAM** do *Controller*. Nos gráficos é assinalado com uma "bolinha"os momentos em que termina o tempo do intervalo de monitorização.

5 MANUAL DE UTILIZAÇÃO

5.1 MÓDULO 1

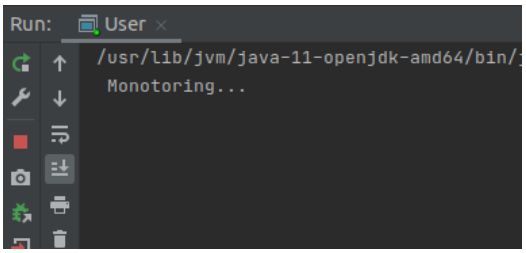

Figura 2: Monitorização a decorrer.

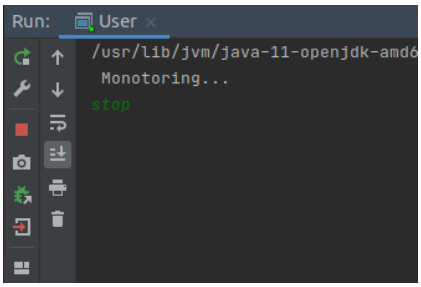

Figura 3: Parar a monitorização.

5.1.1 RESULTADOS

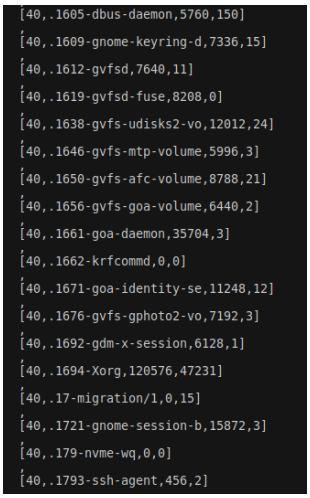

Figura 4: Parte de um *log file*.

Nota: apercebi-me perto do término do trabalho de uma intenção de o *log file* ser *user friendly*, o que faria todo o sentido. Porém, como já tinha feito todo o resto do trabalho baseado nesta formatação do ficheiro, adicionei a seguinte linha ao início dos ficheiros (e ignorei-a no processamento dos dados).

```
INFO: [ SECONDS , PROCESS , RAM , CPU],
[0,.1-systemd,11940,213]
,
[0,.10-ksoftirqd/0,0,18]
```

Figura 5: Início de um log file.

5.2 MÓDULO 2

Figura 6: Janela inicial para escolha do log file.

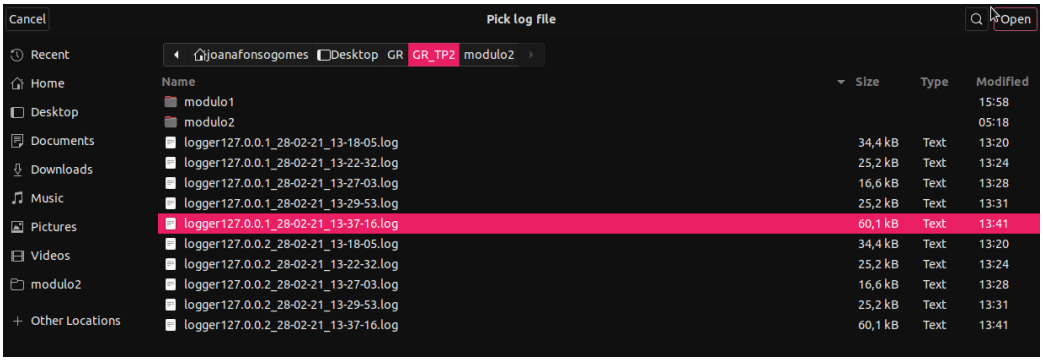

Figura 7: *File Chooser* para selecionar o logfile.

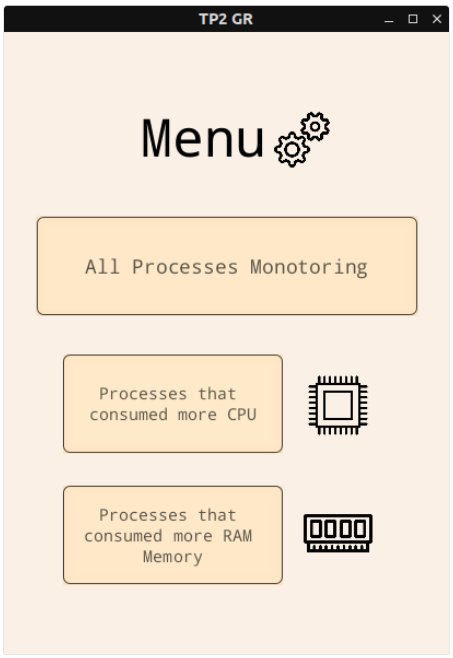

Figura 8: Menu da aplicação.

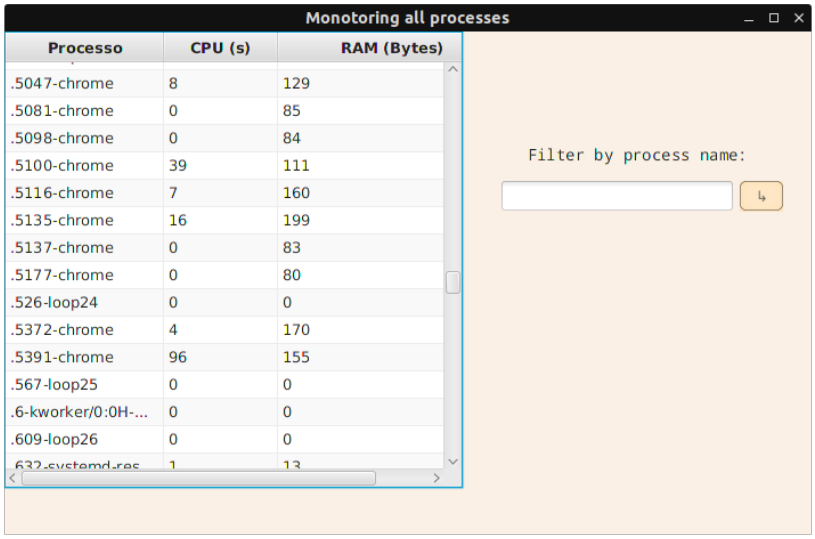

Figura 9: Exemplo de tabela gerada com dados do CPU e de RAM dos processos.

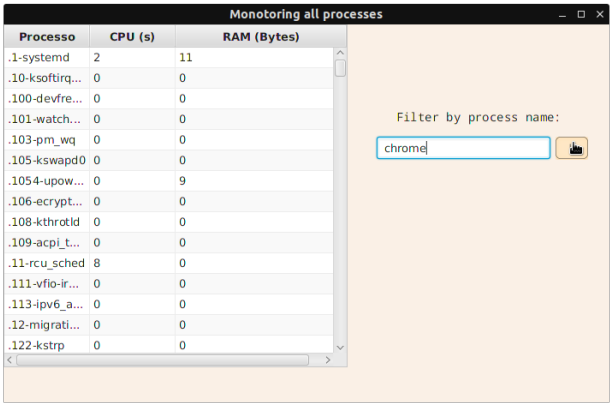

Figura 10: Pesquisa por nome de processo.

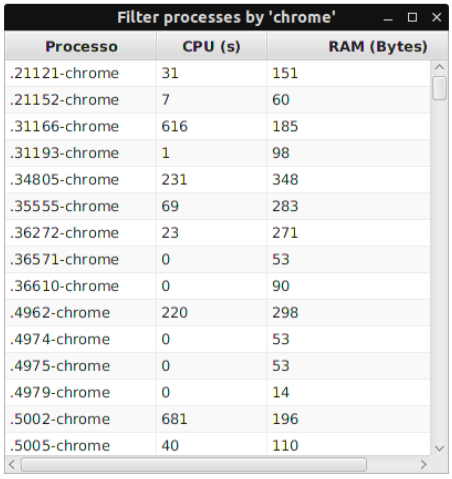

Figura 11: Tabela com os resultados filtrados.

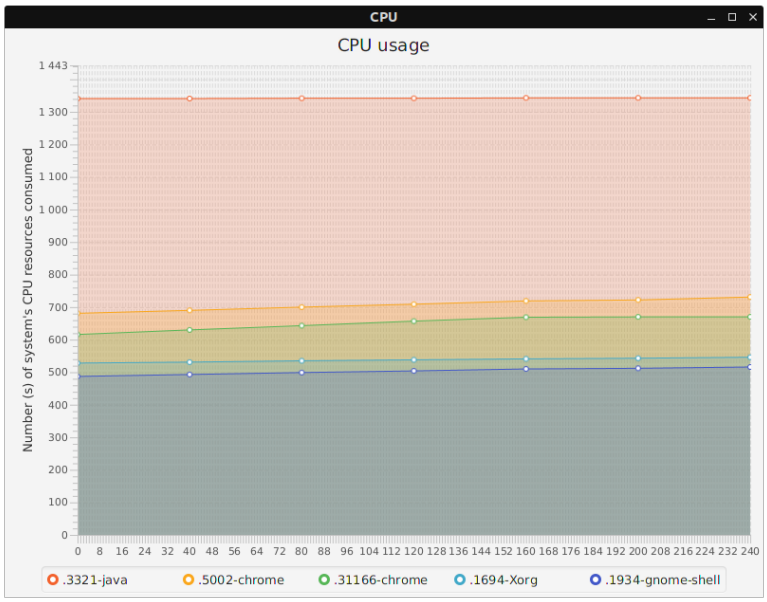

Figura 12: Exemplo de um gráfico com os 5 processos que atingiram os maiores valores de utilização do CPU.

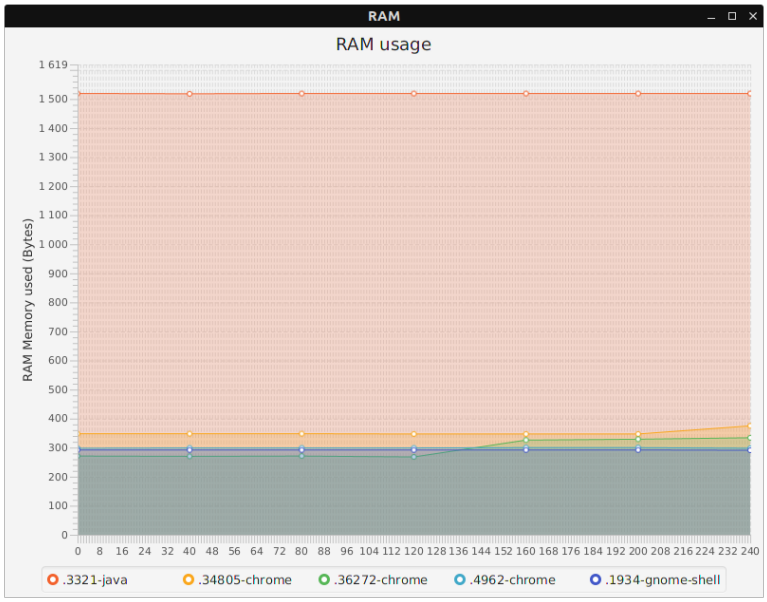

Figura 13: Exemplo de um gráfico com os 5 processos que atingiram maiores valores de utilização de Memória RAM.

6 CONCLUSÃO

Com este projeto foi possível aprofundar conceitos como SNMP e MIBs e obter conhecimentos práticos no que toca a saber aplicar APIs SNMP para construção de ferramentas de monitorização.

Paralelamente a conceitos mais teóricos necessários para o desenvolvimento deste projeto, a criação desta ferramenta, optando pela linguagem *Java*, levou à familiarização com a API *SNMP4J*.

Considerei um projeto bastante interessante, com os seus desafios, e concluo um desfecho positivo até no que toca à referida aprendizagem de conceitos práticos do Simple Network Management Protocol.

Referências

[1] *ManageEngine Pitstop*. URL: https://pitstop.manageengine.com/portal/en/home.

[2] *Oracle JavaFX Documentation Home*. URL: https://docs.oracle.com/javafx/2/ui_controls/table-view.htm](https://docs.oracle.com/javafx/2/ui_controls/table-view.htm.

[3] *SNMP Walk Example using SNMP4J*. URL: https://examples.javacodegeeks.com/enterprise-java/snmp4j/snmp-walk-example-using-snmp4j/.