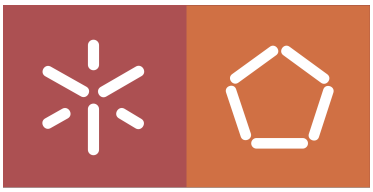

Universidade do Minho
Escola de Engenharia

Gestão de Redes

Trabalho Prático nº2

Mestrado integrado em Engenharia Informática
4º Ano - 1º Semestre

A85308 Filipe Miguel Teixeira Freitas Guimarães

Braga,
28 de fevereiro de 2021

Conteúdo

1 Introdução

Neste relatório encontra-se uma descrição detalhada não só dos resultados obtidos como também do processo em si da realização do trabalho prático nº2 da Unidade curricular de Gestão de redes.

Como proposto pela equipa docente, este projeto tem como tema central a criação de um programa capaz de monitorizar processos a correr num sistema da rede local.

Como estrutura deste relatório, primeiramente apresenta-se os a interpretação de requisitos do enunciado de uma forma sucinta. Posteriormente é apresentada a estrutura do programa e a respetiva implementação e recurso a API's por módulo. Contém também uma secção com o manual de utilização dos diversos módulos do programa.

2 Análise de Requisitos

Como forma de melhor contextualizar o problema, resume-se de seguida os objetivos, restrições e pontos principais do enunciado.

O objetivo principal é permitir monitorizar um ou vários sistemas na rede local. Para isso é proposto que o programa tenha 3 módulos.

2.1 Módulo 1

Monitorização do sistema.

Input

Fornecer ao utilizador uma forma de controlo sobre o módulo para poder introduzir:

- Um ou mais endereços de sistemas a monitorizar.
- Intervalo de monitorização.

Processamento

Realizar as seguintes operações constantemente com a frequência indicada:

- Solicitar ao agente *SNMP* todos os processos a correr na maquina bem como o tempo de *cpu* e memória ocupada.
- Calcular a percentagem de *cpu* e de memória ocupada por cada processo.

Output

- Recolher os dados fornecidos e acrescentar ao ficheiro de *logs* a informação obtida.

2.2 Módulo 2

Analisar os dados obtidos quer seja em tempo real ou histórico.

Input

- Ficheiro de *logs* gerado pelo módulo 1.

Processamento

- Tratar os dados presentes no ficheiro.

Output

- Apresentar aos administradores os dados agregados.
- Gráficos de histórico de utilização de cpu e memória.
- Pesquisa por nome.
- Informações sobre cada processo a correr.

2.3 Módulo 3

Gerar alarmes em tempo real.

Input

- Ficheiro de *logs* gerado pelo módulo 1.
- *Threshold* para cpu e memória.
- (opcional) Email para enviar os alarmes.

Processamento

- Analisa os *logs* fornecidos e verifica se estão acima do threshold.

Output

- Escreve num ficheiro os alarmes gerados.
- (opcional) Envia um email com os alarmes gerados.

3 Resolução

Após a análise de requisitos, nesta secção, é abordado todo o processo de resolução do problema proposto. Para isso apresenta-se na figura 1 o diagrama que divide o programa em módulo e estabelece o meio de comunicação entre eles, neste caso o ficheiro de *logs*.

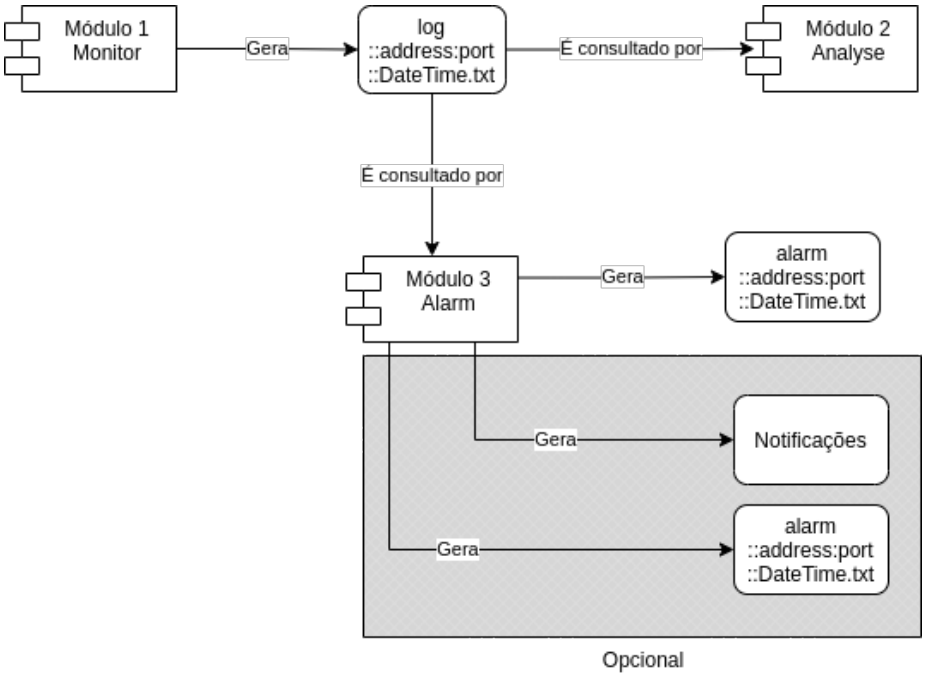

Figura 1: Diagrama de módulos do programa

Com os requisitos apresentados escolhi desenvolver este projeto em java recorrendo à API SNMP4J.

De forma a explicar melhor o funcionamento de cada módulo esta secção é dividida pelos diferentes módulos e no final são analisados os resultados. Como todo o código está documentado será apenas fornecida uma breve explicação do funcionamento dos métodos em cada classe.

3.1 Módulo 1 - Monitor

3.1.1 VARIABLES

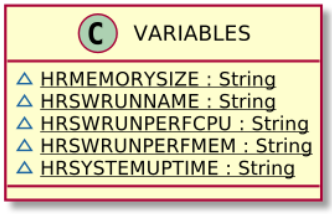

Figura 2: Classe Variables.

Nesta classe encontra-se todos os OID's das MIB's utilizadas. Para atingir os objetivos propostos decidi usar as seguintes:

- **HRSYSTEMUPTIME = ".1.3.6.1.2.1.25.1.1.0"** - Para obter o uptime do sistema-
- **HRSWRUNNAME = ".1.3.6.1.2.1.25.4.2.1.2"** - Para obter o nome dos programas a correr bem como retirar o PID que é igual ao índice.
- **HRSWRUNPERFCPU = ".1.3.6.1.2.1.25.5.1.1.1"** - Para obter o CPU dos programas a correr.
- **HRSWRUNPERFMEM = ".1.3.6.1.2.1.25.5.1.1.2"** -Para obter a memória dos programas a correr.
- **HRMEMORYSIZE = ".1.3.6.1.2.1.25.2.2.0"** - Para obter a memória RAM total.

3.1.2 Log

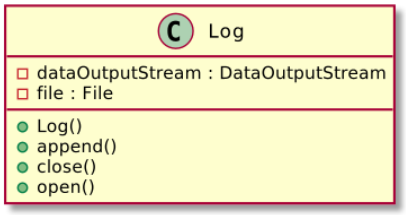

Figura 3: Classe Log.

Responsável pela criação e escrita do ficheiro de log's. O construtor cria um ficheiro com um nome que contem o endereço e porta que está a ser monitorizado e a data e hora de criação. Fornece métodos como o *append* que usando o *DataOutputStream* adiciona a string fornecida como argumento ao ficheiro. Um excerto exemplo do ouput desta classe seria:

```
...
{ "process" = {"pid":852, "name":"compton", "mem":0.46518737, "cpu":2.6666667} }
{ "process" = {"pid":904, "name":"dockerd", "mem":0.72551703, "cpu":0.0} }
{ "process" = {"pid":913, "name":"containerd", "mem":0.3943785, "cpu":0.033333335} }
...
```

3.1.3 Process

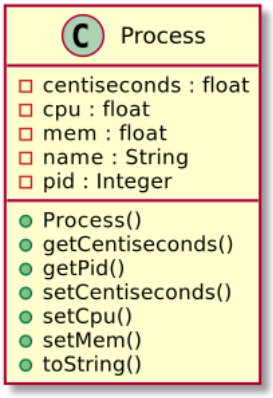

Figura 4: Classe Process.

Classe que contem a estrutura para um objeto Processo que representa um processo a correr no computador. Este já é preenchido com ambas as percentagens (cpu e ram) enviando os dados necessários aos *setters* implementados.

3.1.4 Monitor

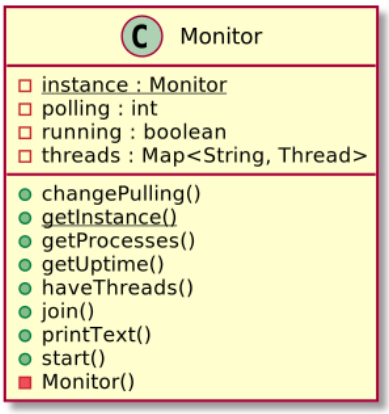

Figura 5: Classe Monitor.

Classe que contem o "cérebro"da operação. Mantém uma instância estática que é partilhada por todas as outras classes de forma a que haja apenas um fonte de informação. Armazena também todas as threads que é responsável por manter ou eliminar a pedido. Fornece métodos para mudar o tempo de poll bem como começar uma nova monitorização. É aqui que os dados são compilados e enviados ao log para serem escritos no ficheiro.

3.1.5 Cliente

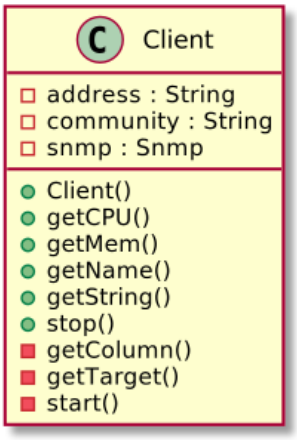

Figura 6: Classe Client.

Classe usada para comunicação com o SNMP. Foi desenvolvida partindo da explicação do uso do snmp4j encontrada no blog *jayway*[1].

O diagrama seguinte explica o funcionamento genérico do pedido de uma string ou tabela ao agente.

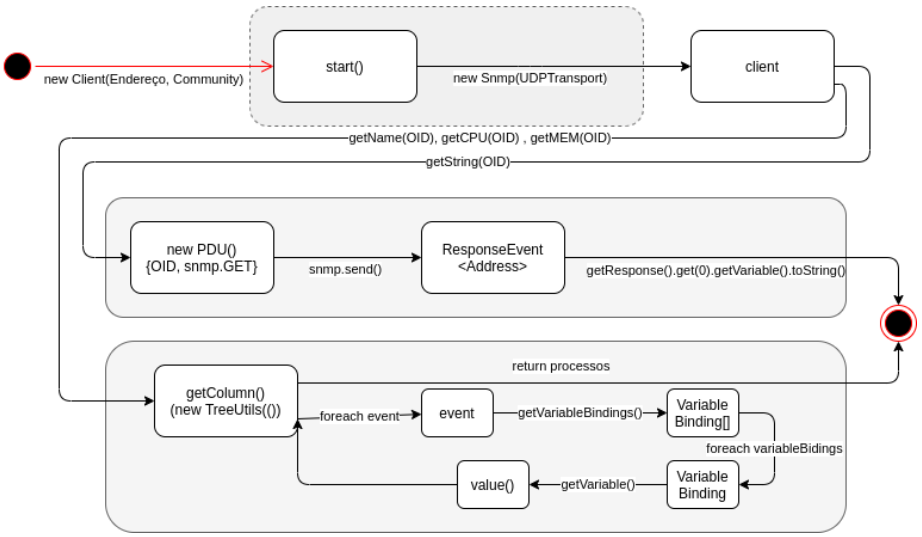

Figura 7: Funcionamento do cliente.

3.2 Módulo 2 - Analyse

3.2.1 Log

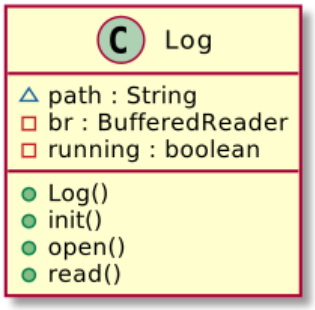

Figura 8: Classe log.

Classe para ler do ficheiro log gerado pelo módulo 1. Primeiramente recorre-se ao método init() ue lê linha a linha e, posteriormente, ao read() que lê em tempo real o ficheiro e fica à espera (num wait() de 10 segundo de cada vez) por mais linhas. Esta classe é responsavel por adicionar estados ao agregador usando como auxiliar a classe Tradutor para transformar linhas em Processos.

3.2.2 Tradutor

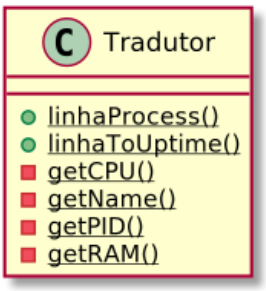

Figura 9: Classe Tradutor.

Classe auxiliar para traduzir os campos de cada linha do ficheiro de log's num processo recorrendo a expressões regulares para analisar os padrões.

3.2.3 Process

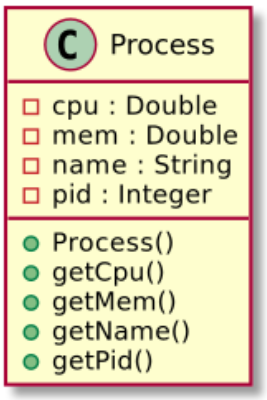

Figura 10: Classe Process.

Classe que contem a estrutura de um processo e não oferece nenhum método "especial".

3.2.4 ProcessGroup

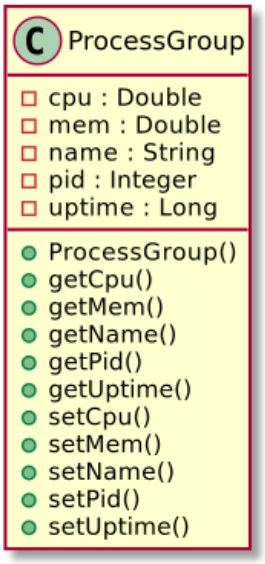

Figura 11: Classe ProcessGroup.

Classe que varia da anterior por representar um grupo de processos com o mesmo PID para o calculo de RAM e CPU ao longo do tempo.

3.2.5 Estado

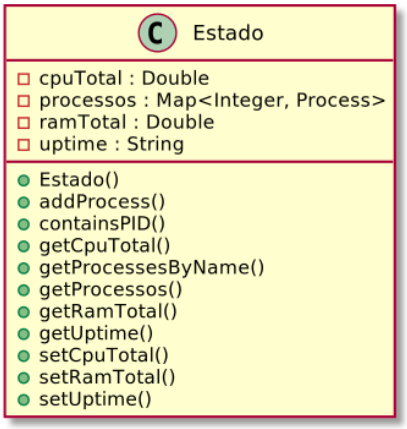

Figura 12: Funcionamento do cliente.

Classe que representa o estado do computador num determinado momento (uptime). Contem o método getProcessesByName() que implementa uma pesquisa por nome de processo.

3.2.6 Agregador

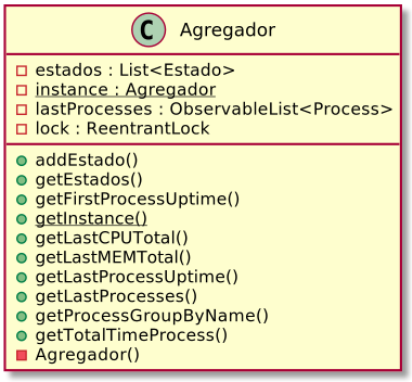

Figura 13: Classe Agregador.

Classe que contem o "cérebro"da operação. Tal como no módulo 1, este módulo contem também esta classe que contem apenas uma instância por execução. Armazena uma lista de estados e mantém uma lista atualizada de processos a correr, de forma a que a interface atualize de forma automática. Contém também vários métodos para apresentar a informação nas estrutura de memória para a interface gráfica.

3.3 Módulo 3 - Alarm

3.3.1 Tradutor

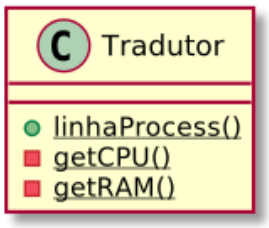

Figura 14: Classe Agregador.

Classe auxiliar para traduzir os campos de cada linha do ficheiro de log's num processo recorrendo a expressões regulares para analisar os padrões.

3.3.2 Process

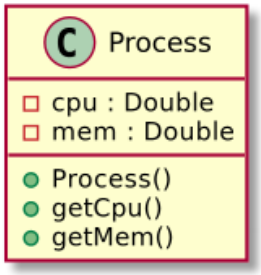

Figura 15: Classe Agregador.

Classe que contem a estrutura de um processo simplificada para só armazenar a memoria e cpu usados e não oferece nenhum método "especial".

3.3.3 Email

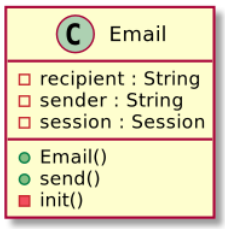

Figura 16: Classe Agregador.

Classe que envia emails a partir do email *gr.gvr.2021@gmail.com*. Oferece o construtor que aceita o email de destino e o método ***send()*** que irá enviar o alarme caso seja solicitado. Faz uso do *javax mail* para fornecer este serviço.

3.3.4 Alarm

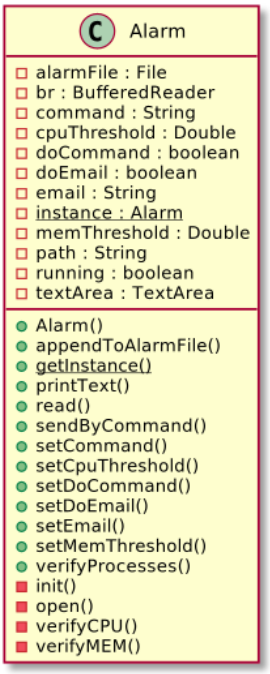

Figura 17: Classe Agregador.

Classe principal do programa. Armazena todas as definições introduzidas na interface gráfica e oferece os métodos para verificar se existe alarmes e enviar.

Difere no módulo anterior na parte de ler o ficheiro em só ler e manter as informações referentes ao ultimo estado acrescentado ao ficheiro.

Quando é gerado um alarme é também acrescentado (se não existir é criado o ficheiro) ao ficheiro de alarmes uma linha representativa de qual threshold foi ultrapassado.

3.4 Análise de Resultados

Estou muito satisfeito com os resultados obtidos. Penso que foi cumprido todos os objetivos propostos.

No módulo 1 seria interessante, além de fornecer a interface gráfica de configuração ao utilizador, fornecer um ficheiro que pudesse ser modificado, para não obrigar o utilizador a interagir sempre com a interface gráfica.

Quanto ao módulo 2 seria interessante criar *Data Access Objects* para uma base de dados de forma a que os dados não ficassem permanentemente em memoria e, desta forma, dar mais estabilidade ao programa

Como no módulo 1, no módulo 3 seria também interessante fornecer um ficheiro para configuração.

4 Manual de Utilização

Como foi referido anteriormente este projeto está dividido em três módulos distintos e, por conseguinte, existem três programas para executar. Estes encontram-se na pasta *Jars* ou podem ser compilados usando o comando maven **$ mvn clean package**.

4.1 Módulo 1 - *Monitor*

Ao abrir o primeiro módulo temos a página principal (figura 18).

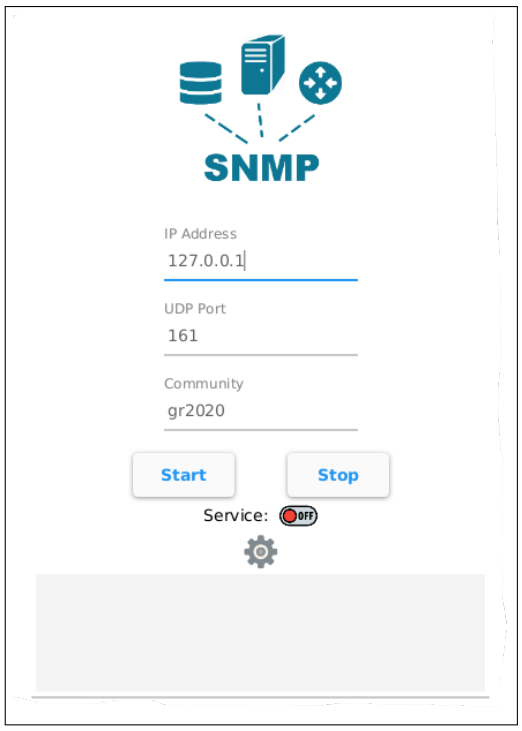

Figura 18: Página principal.

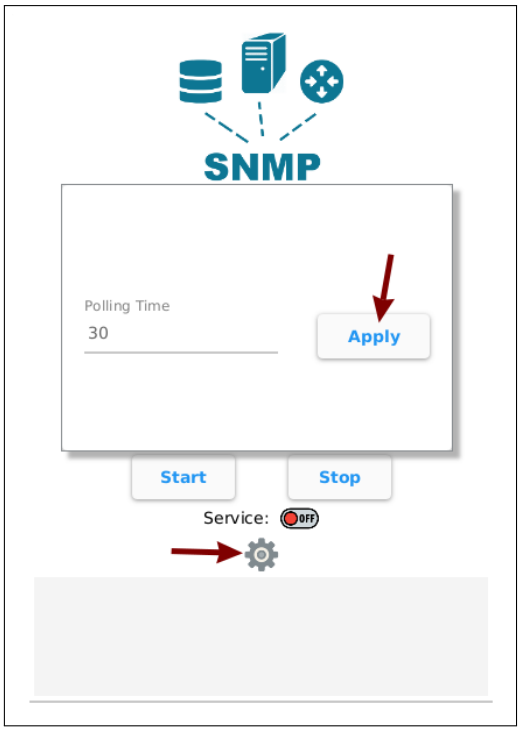

Figura 19: Definições.

Para aceder às definições e alterar o tempo de *polling* carregamos na roda dentada como se pode ver na figura 19. Este tempo pode ser alterado a qualquer momento no decorrer da execução e terá efeito no próximo polling.

Para começar a monitorização têm de se preencher todos os espaços mostrados no ecrã e clicar em *Start* como se pode ver no exemplo dado na figura 20. Quando o serviço está a executar conseguimos ver que o símbolo (onde a seta azul aponta) fica verde e fica *on*.

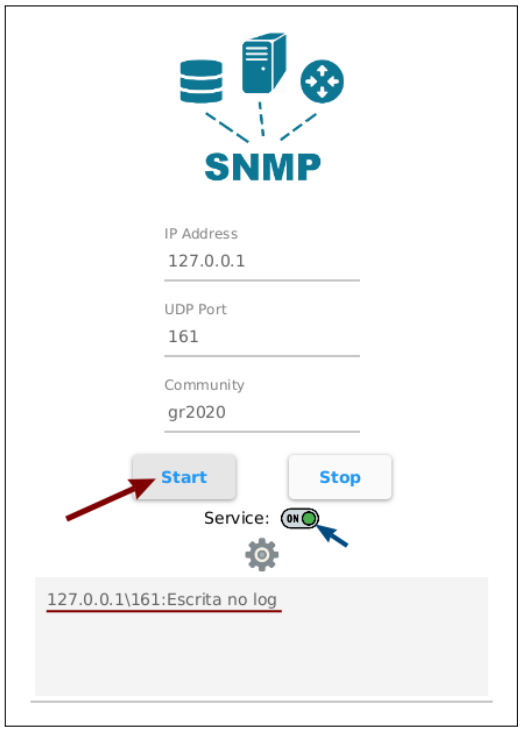

Figura 20: Start.

Figura 21: Stop.

Para terminar a monitorização de um determinado *host*, primeiro introduz-se os campos todos e clica-se em *Stop* como se pode ver na figura 21. Se não existir mais nenhum *host* ativo o serviço (seta azul) fica vermelho com indicação *off*.

Quando fechamos a aplicação todos os *host* param de ser monitorizados.

4.2 Módulo 2 - *Analyse*

Ao executar este módulo temos a página de seleção do ficheiro gerado pelo monitor como na figura 22.

Figura 22: Choose File.

Figura 23: Start.

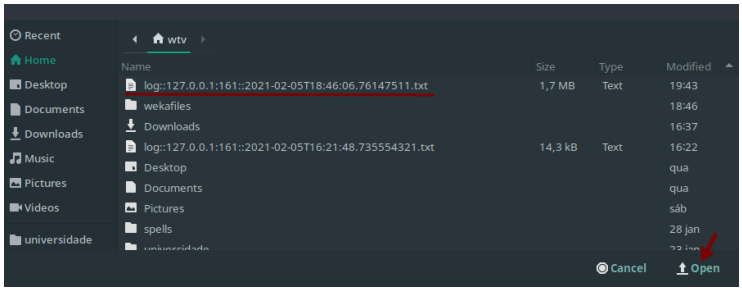

Figura 24: Janela de escolha do ficheiro (depende do sistema operativo)

Após carregar para escolher o ficheiro, é preciso escolher o ficheiro certo gerado pelo módulo 1 (como se vê na figura 24). Depois de escolhido o ficheiro o botão *start* fica disponível e pode-se começar a analisar os dados.

Depois do processo de escolha do ficheiro aparece a página principal da aplicação (figura 25) com os respetivos botões/informações:

1. Mostra o gráfico com a ocupação do cpu ao longo do tempo.
2. Mostra o gráfico com a ocupação da memória ao longo do tempo.
3. Mostra mais informações sobre os processos. Permite a pesquisa por nome.
4. Sair da aplicação
5. Tabela dos processos a correr atualmente.
6. Velocímetro com percentagem de cpu no momento.
7. Velocímetro com percentagem de memória no momento.
8. Informações sobre o processo selecionado.

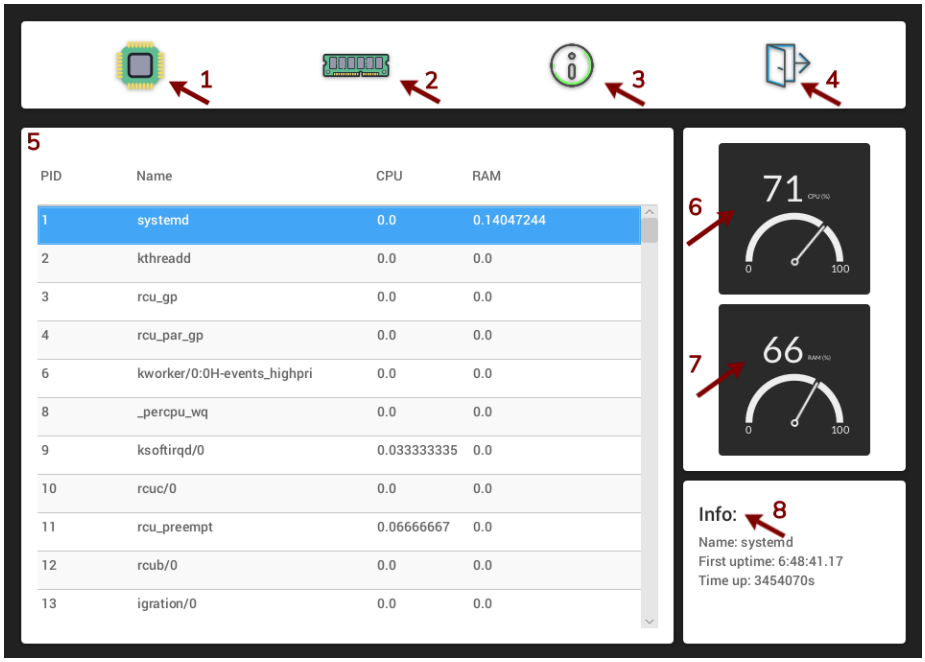

Figura 25: Página principal com anotações.

Tanto o ponto 1 como o ponto 2 apresentam um gráfico como o mostrado na figura 26.

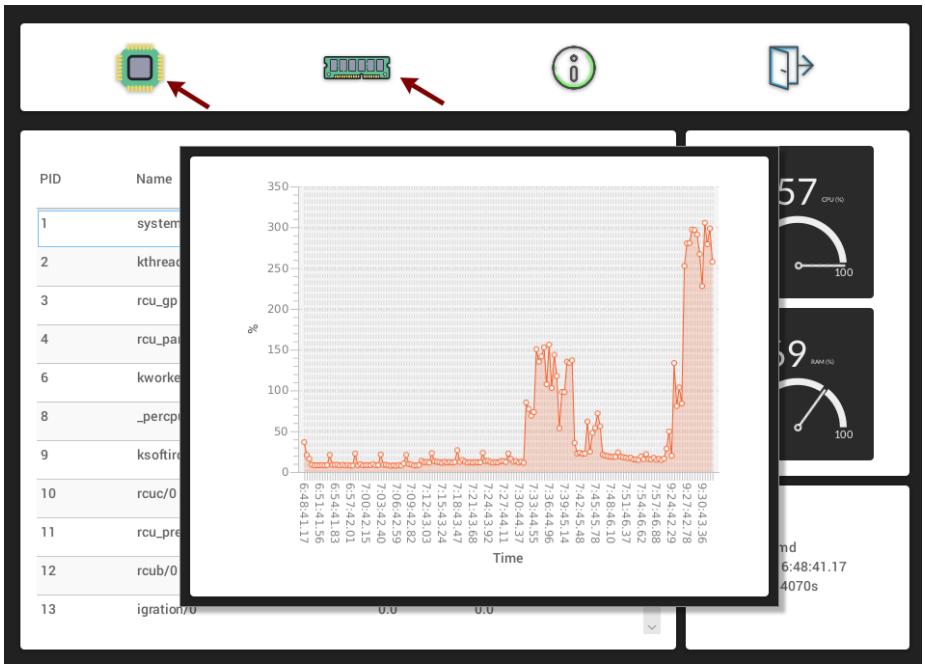

Figura 26: Exemplo de gráfico mostrado pelas opções apontadas.

Ao entrar no ponto 3 podemos inserir o nome do processo que queremos observar, ou podemos optar por deixar vazio e ver para todos os processos.

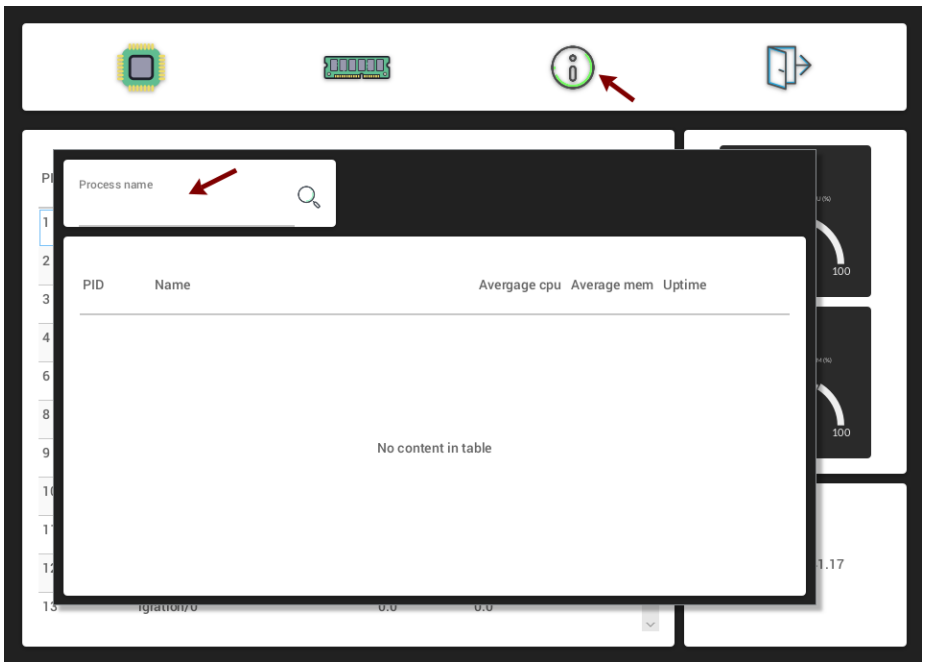

Figura 27: Página info.

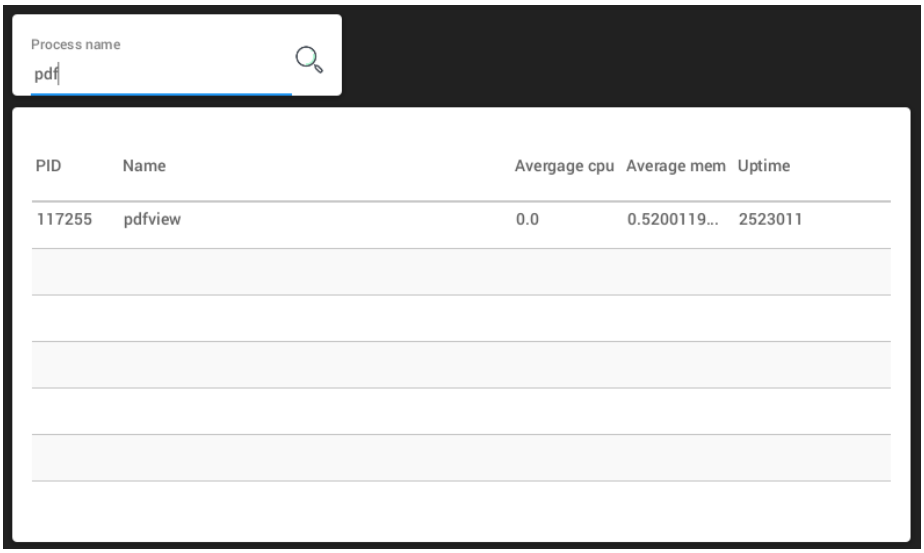

Figura 28: Pesquisa.

4.3 Módulo 3 - *Alarm*

A pagina inicial deste módulo é a apresentada na figura seguinte.

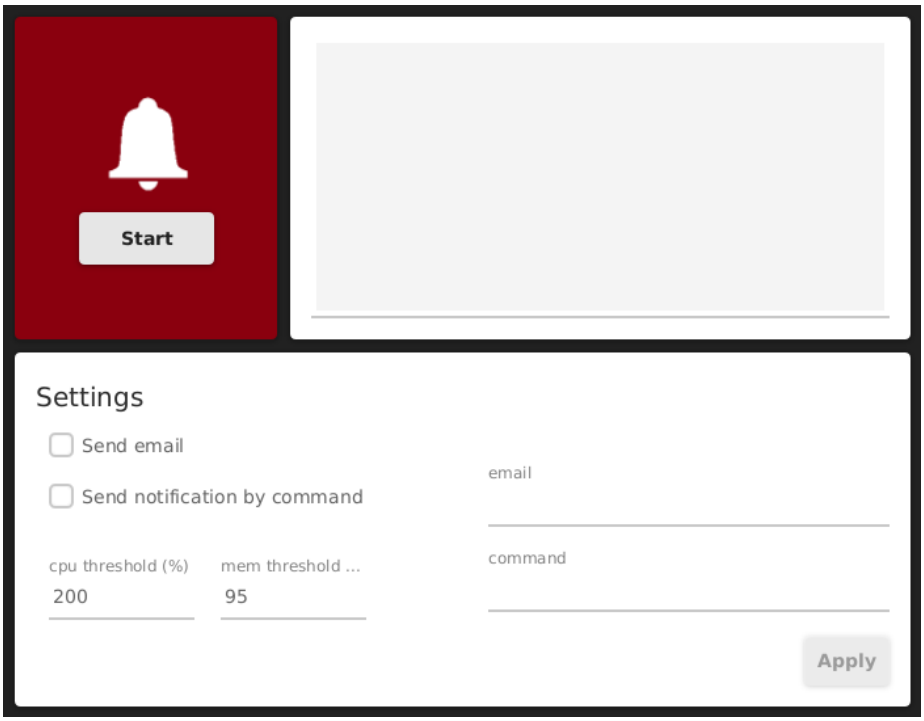

Figura 29: Página inicial do módulo 3.

Ao clicar em *start* aparecerá a janela para selecionar o ficheiro pretendido, à semelhança com o módulo 2 tem de se escolher o ficheiro que está a ser gerado pelo monitor.

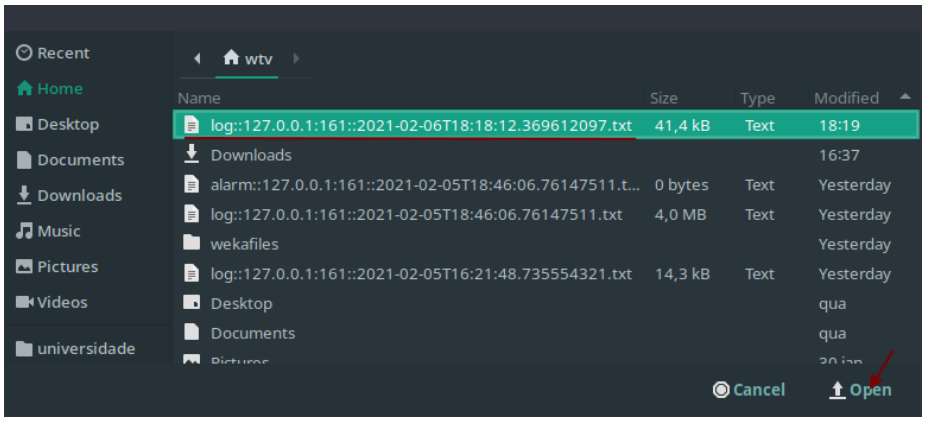

Figura 30: Escolha do ficheiro.

Com o ficheiro selecionado consegue-se ver o output como sublinhado na figura 31.

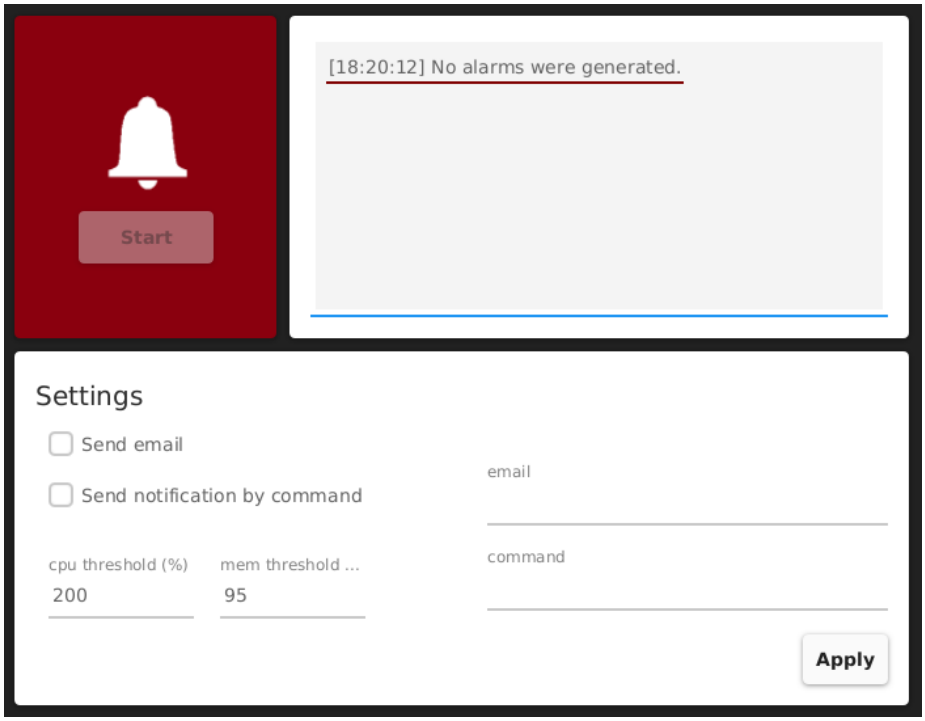

Figura 31: Módulo 3 a funcionar.

Podem-se mudar as definições como pretendido (exemplo na figura 32) e após clicar em *Aplly* aparecerá a confirmação.

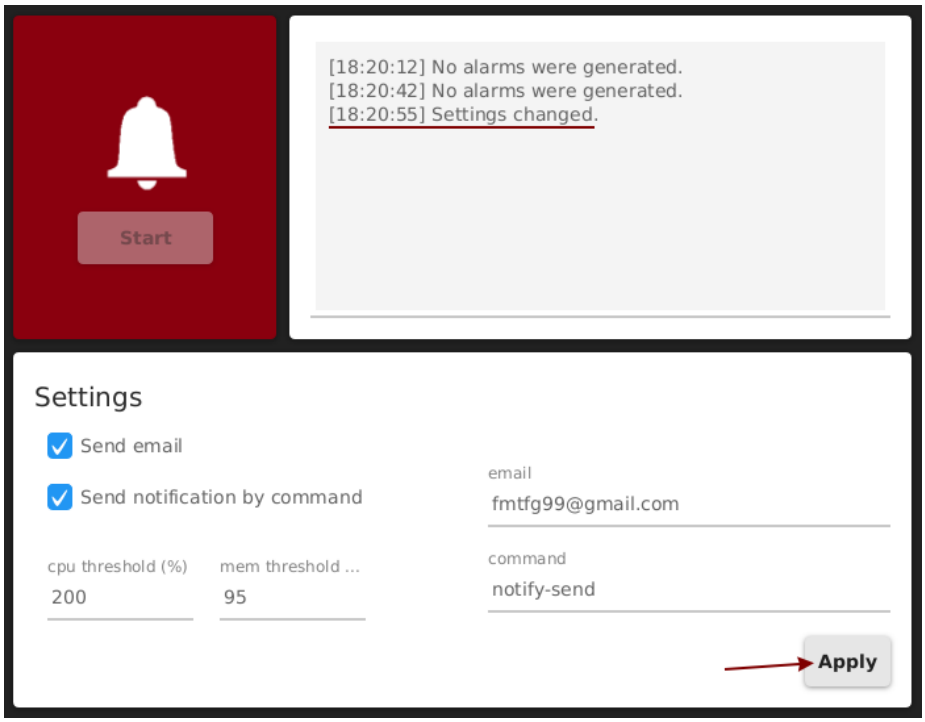

Figura 32: Alterar definições

Caso o alarme seja ativado aparecerá conforme descrito nas definições. Neste exemplo conseguimos ver tanto na aplicação como no sistema operativo e no email (como se pode ver nas figuras abaixo).

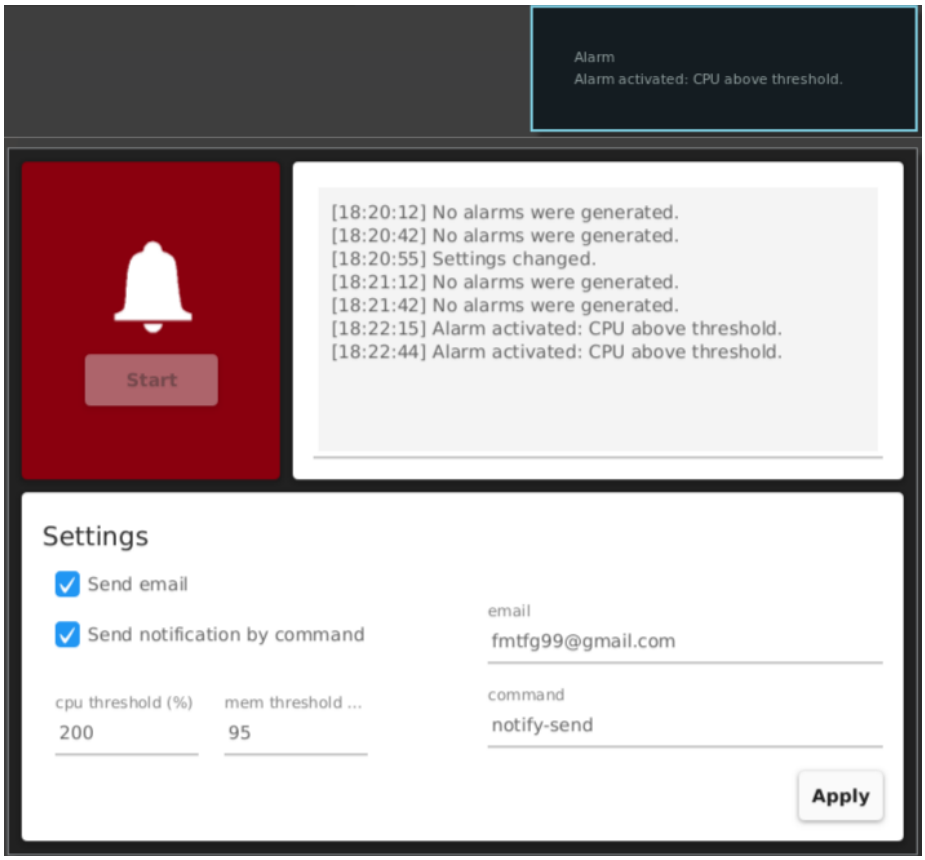

Figura 33: Notificações no sistema operativo.

Figura 34: Email enviado.

5 Conclusão

Neste projeto é apresentado o conceito de gestores SNMP. Ao realizar esta ficha prática ganhei muitos conhecimentos na API SNMP4j que me forneceu ferramentas práticas para compreender melhor o protocolo SNMP.

Referências

[1] Johan Rask. *Introduction to snmp4j*. URL: https://blog.jayway.com/2010/05/21/introduction-to-snmp4j/. (accessed: 20.12.2020).