

Gestão de Redes

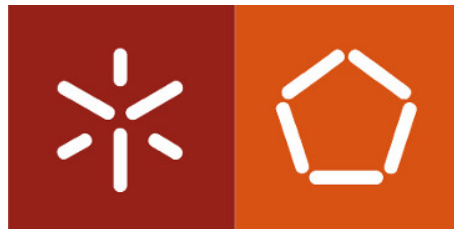
28 de Fevereiro de 2021

Trabalho Prático 2

a84485

Tiago Magalhães

Ferramenta de monitorização



Mestrado Integrado em Engenharia Informática
Universidade do Minho

Conteúdo

1	Introdução	2
2	Concepção/Desenho da resolução	3
2.1	Descrição da arquitetura	3
2.2	Monitorização	4
2.2.1	MIB utilizada para recolhas do dados de gestão	4
2.2.2	Politica de <i>logs</i>	5
2.2.3	Sintaxe ficheiro de <i>logs</i>	6
2.2.4	Ficheiro de configuração	6
2.3	API Server	7
2.4	Notificações	7
2.5	App Server	8
3	Conclusão	11

1 Introdução

Neste trabalho prático, o objetivo era criar um programa para monitorização e análise de utilização dos recursos do sistema local pelos processos ativos num qualquer *host*.

Nas próximas secções, serão explicados cada um dos passos para a conceção deste projeto e para a sua estruturação.

Pode-se encontrar o código num repositório [Github](#).

2 Concepção/Desenho da resolução

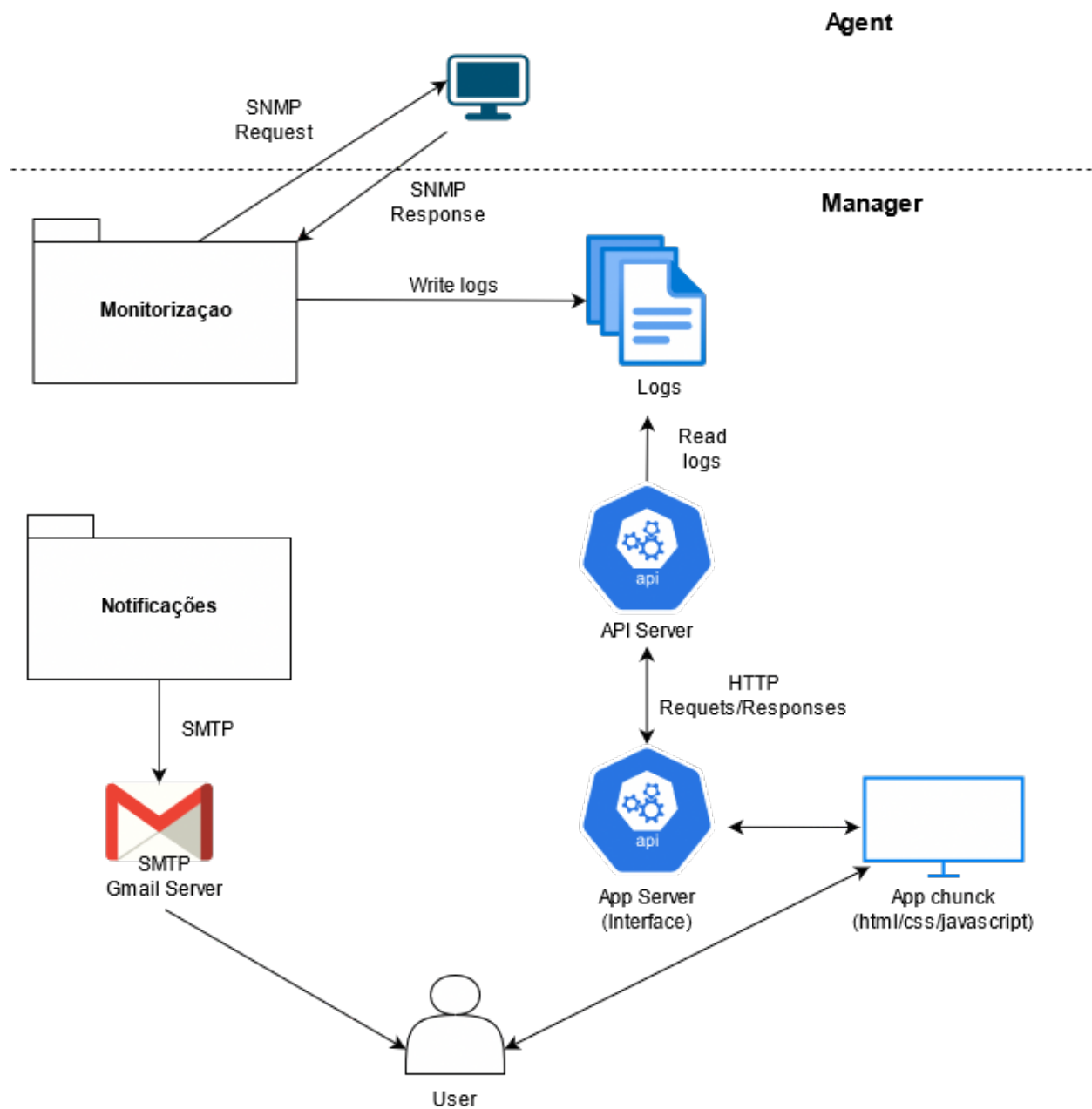


Figura 1: Arquitetura da ferramenta de monitorização

2.1 Descrição da arquitetura

Através do modelo do sistema podemos observar que existem 4 componentes principais sendo estas:

- **Monitorização:** Módulo responsável por gerar os *logs* com os dados de gestão dos recursos de CPU e memória RAM por parte dos processos ativos no *host*.
- **API Server:** API que a partir dos dados presentes nos *logs* gerados pelo módulo de monitorização irá fornecer informação tais como: número de processos que consumiram mais percentagem de memória num *host*, quais os processos que tiveram uma duração mais longa, etc.
- **App Server:** Servidor que fornece páginas dinâmicas ao cliente e que consome dados do API Server.
- **Notificações:** Módulo responsável por gerar mensagens de email com o alarme.

2.2 Monitorização

A tecnologia utilizada neste módulo é *python* e uma vez que os dados sobre a gestão de recursos são obtidos através do protocolo SNMP foi utilizada a biblioteca Easy SNMP que possui uma boa documentação, manutenção e é mais rápida em relação a outras para a mesma linguagem.

Neste módulo existem duas classes, a classe monitorworker e a classe monitor que atua como um *wrapper* da classe monitorworker, assim sendo a classe monitor percorre um ficheiro de configuração e para cada *host* cria uma *thread* com a classe monitorworker que irá recolher os dados sobre gestão de recursos de CPU e memória RAM e assim gerar os ficheiros de *log*.

2.2.1 MIB utilizada para recolhas do dados de gestão

A MIB utilizada para recolher dados acerca de CPU e memória RAM foi a *Host Resources MIB*[1].

Objetos da MIB considerados:

- HRSWRUNTABLE - '.1.3.6.1.2.1.25.4.2.1.2' - Tabela com os processos ativos no *host*;
- HRSWRUNPERFCPU - '.1.3.6.1.2.1.25.5.1.1.1.' + 'índice_processo' - Objecto com número total de centi-segundos do total de recursos de CPU do sistema consumidos por um processo;
- HRSTORAGESIZE - '.1.3.6.1.2.1.25.2.2.0.' - Objecto com número total de memória RAM do *host*;
- HRSWRUNPERFMEM - '.1.3.6.1.2.1.25.5.1.1.2.' + 'índice_processo' - Objecto com número total de memória alocada ao processo.
- HRPROCESSORLOAD = '.1.3.6.1.2.1.25.3.3.1.2' - Tabela com médias de percentagens de tempos que cada processador esteve ocupado no último minuto;
- HRSYSTEMUPTIME - '.1.3.6.1.2.1.25.1.1.0' - Quantidade de tempo desde que *host* foi inicializado.

2.2.2 Política de *logs*

Quanto aos *logs*, estes irão ficar armazenados dentro da diretoria *logs*, de modo a distinguir a que *host* os *logs* se referem existe uma diretoria identificada pelo endereço IP do *host* ou caso seja a rede do utilizador esta é identificada por *localhost*, onde se fará a distinção de *logs* sobre memória RAM e CPU em duas diretorias com nome CPU e RAM. Dentro de cada uma destas ficarão os *logs* armazenados em ficheiros de texto com o nome do processo correspondente, sendo que os nomes dos processos que possuam caracteres inválidos para nome de ficheiro, os caracteres são eliminados.

Existe também um ficheiro *cpugeral* que tem informação com as percentagens de tempos que cada processador esteve ocupado no último minuto.

Exemplo da estrutura:

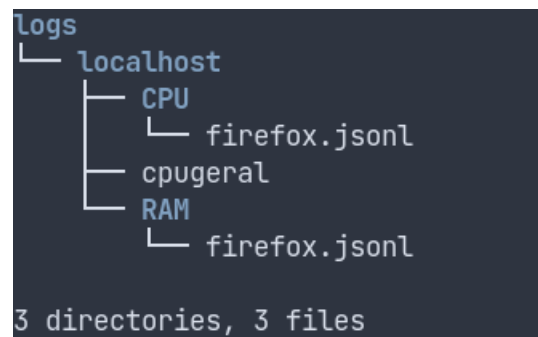


Figura 2: Estrutura de armazenamento *logs*

2.2.3 Sintaxe ficheiro de *logs*

Para a sintaxe de cada *log* são geradas *Jsonlines*¹, uma vez que têm uma estrutura de dicionário que permite que sejam facilmente manipuláveis em *python*, que tem como principal estrutura de dados o dicionário. Dado que serão geradas grandes quantidades de dados, já que estamos a monitorizar ativamente, este é um formato conveniente para processar linha a linha, não sendo necessária muita memória. Consequentemente é utilizada a biblioteca *jsonlines* para facilitar a manipulação destas.

Exemplo:

```
// Exemplo ram
{"time": "2021-02-27 01:13:21.269205", "mem": 3.1258911223894272}
// Exemplo cpu
{"time": "153956", "cpu": 34568, "date": "2021-02-27 01:13:21.269205"}
```

O valor em "mem" é já como percentagem, obtida através da equação:

$$\%memoria = (memoria_alocada / memoria_total) \times 100$$

O valor "time" refere-se ao *system uptime*.

2.2.4 Ficheiro de configuração

Neste módulo existe um ficheiro de configuração que segue o formato de uma estrutura INI².

Sempre que o ficheiro de configuração é atualizado o programa reinicializa, tendo-se usado a biblioteca *pyinotify* que observa mudanças num determinado ficheiro.

Neste ficheiro de configuração existem 4 secções *Hosts*, *Port*, *Community* e *Pooling* em que o utilizador na primeira define o número de hosts que pretende monitorizar (*number*) e os endereços IP de cada um destes seguindo a seguinte sintaxe 'host <número> = <endereço_ip>', na secção *Port* define-se a porta UDP do agente para cada *host*, na secção *Community* define-se as *community string* para se autenticar ao agente e por fim na secção de *Pooling* define-se o tempo de *pooling*.

¹<https://jsonlines.org/>

²Arquivos INI são arquivos de texto simples com uma estrutura básica composta de "secções" e "propriedades".

Exemplo:

```
[Hosts]
number = 1
host.0 = localhost

[Port]
host.0 = 161

[Community]
host.0 = gr2020

[Polling]
host.0 = 30
```

2.3 API Server

A tecnologia utilizada para construir a API foi *python* com recurso à *framework* Flask.

A API gera informação a partir dos dados presentes nos *logs*, mais informação acerca da API pode ser consultada em anexo na pasta docs.

Entre os cálculos mais importantes realizados por esta API, encontra-se a percentagem de CPU utilizada que é calculada através de cada dois intervalos presentes no ficheiro de *logs* em relação ao CPU.

Equação: $cpu_usage = (\Delta cpu / \Delta time) * 100$, em que Δcpu é a diferença dos valores de cpu (número total de centi-segundos do total de recursos de CPU do sistema consumidos por um processo) de cada intervalo, mas convertidos em segundos e $\Delta time$ a diferença dos tempos de recolha de cada valor de CPU.

2.4 Notificações

Este módulo é responsável de gerar mensagens email com o alarme, para isto é utilizado um ficheiro de configuração com estrutura similar ao utilizado no módulo de monitorização no entanto este têm uma secção com o(s) nome(s) do(s) processo(s) a observar e outra com informações das contas de email de onde vai ser enviado o email e para onde vai ser recebido.

Exemplo:

```
[firefox]
host = localhost
type = RAM
threshold = 1.1

[GmailUser]
gmail_user = fusemailtestts@gmail.com
gmail_password = password
```



```
email_to = tiagohomagalhaes@gmail.com
```

Na secção com o nome do processo é identificado um limite(*threshold*), no exemplo é a percentagem de memória e é identificado a que tipo de dado de gestão é referente(CPU ou RAM) e o host onde o processo está presente.

É utilizado o *gmail* como fornecedor, por isso no *gmail_user* e *gmail_password* deve-se colocar informações relativas a uma conta *gmail*. Para não ser necessário usar uma conta *gmail*, teria de se configurar um servidor SMTP que enviasse os emails para as aplicações *gmail*, *outlook*, etc., bem como ter a porta 50(*default* SMTP) aberta.

2.5 App Server

A tecnologia utilizada para construir a aplicação *web* foi NodeJs com recurso à *framework* Express.

Este servidor oferece uma interface *web* interativa com a inclusão de gráficos.

Interface(UI) exemplos:

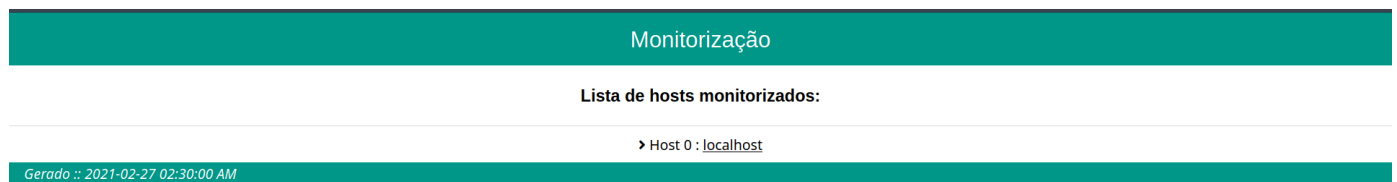


Figura 3: Página inicial.

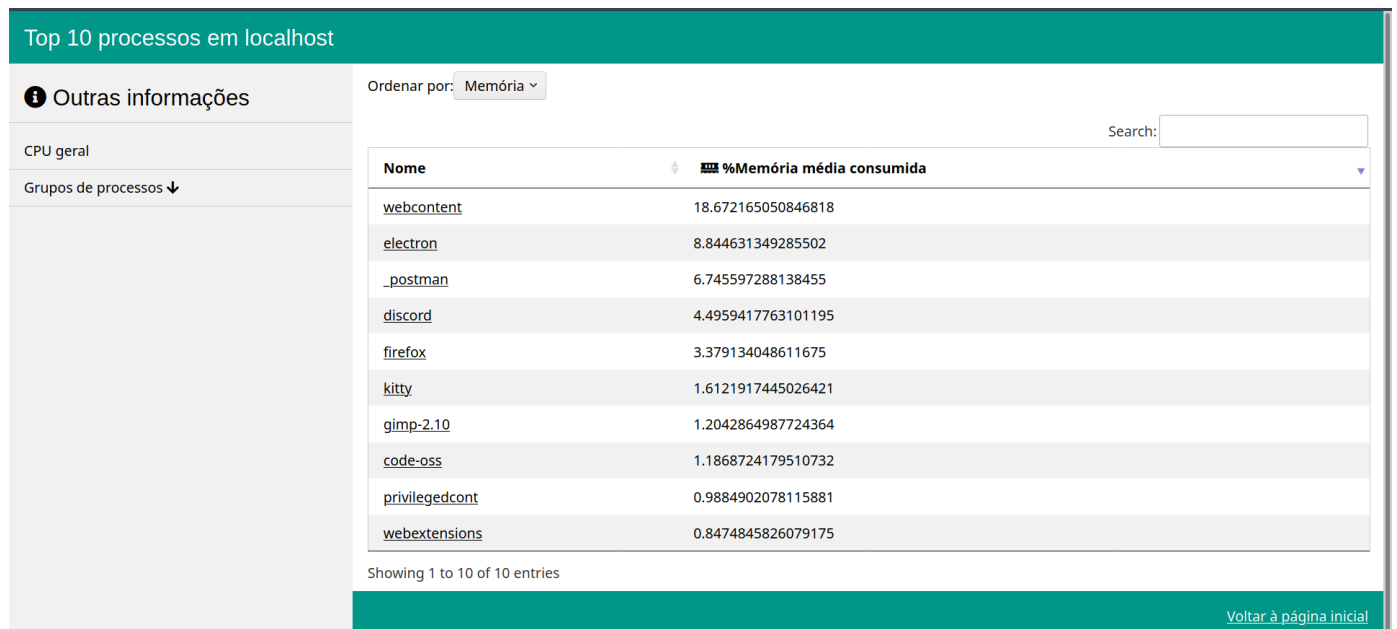


Figura 4: Top processos monitorizados.

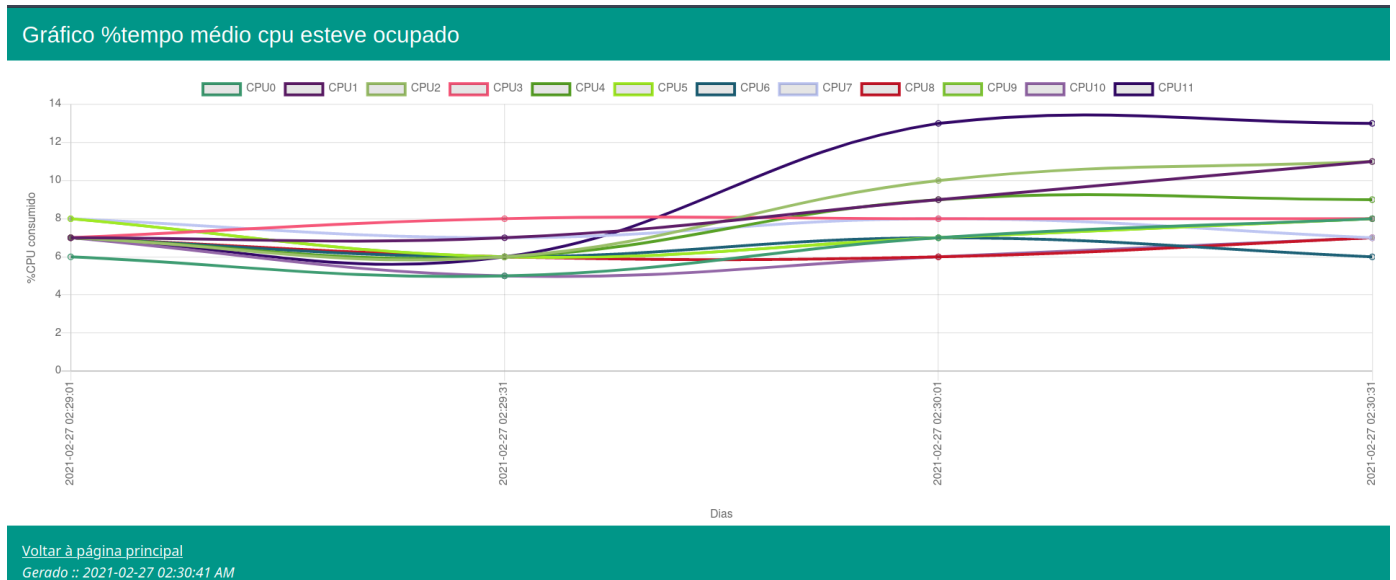


Figura 5: Percentagens médias de cpu não ocupado nos cpus do *host*.

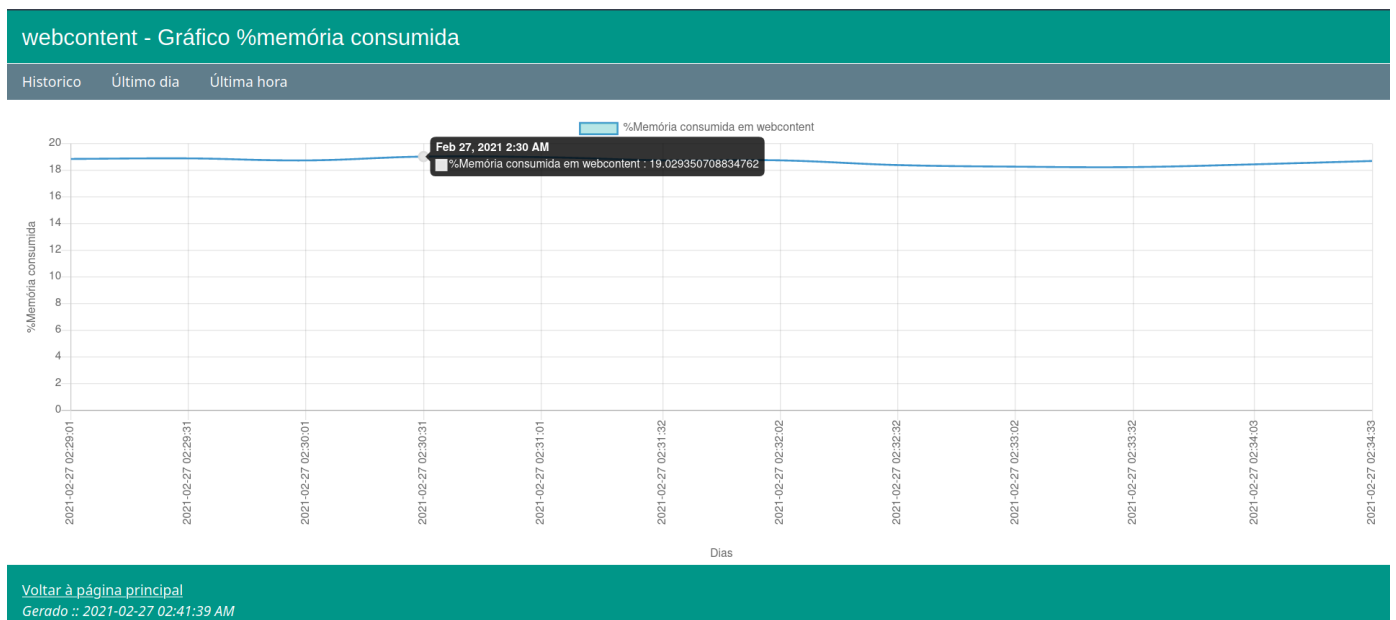


Figura 6: Gráfico de percentagem de memória RAM utilizada por um processo.

Processos de baixa duração(segundos) em localhost

Show10entries

Search:

Nome	Duração (segundos)	Frequência de aparecimento (segundos)
kworker2:2-mm_percpu_wq	30.356967	241.107319
kworkeru24:1-ext4-rsv-conversion	30.356849	105.481949
kworkeru24:2-ext4-rsv-conversion	30.18031	181.11982
kworker10:1-events	30.144978	0
kworkeru24:2-events_unbound	30.143772	100.55342733333333
kworker4:2-mm_percpu_wq	30.133055	100.55705266666666
kworker8:0-mm_percpu_wq	30.132949	135.764099
kworker3:1h-events_highpri	30.13245	67.88488674999999
kworker0:1h-kblockd	30.132049	0
kworker3:1h-kblockd	30.126085	67.88639674999999

Showing 1 to 10 of 36 entries

Previous

1

2

3

4

Next

[Voltar à página principal](#)

Gerado :: 2021-02-27 02:41:47 AM

Figura 7: Processos de pequena duração.

3 Conclusão

Com a realização deste projeto, permitiu consolidar a aprendizagem da UC de Gestão de redes, bem como perceber os principais desafios ao criar uma ferramenta destas no lado do *Manager*.

Concluindo este trabalho permitiu uma maior experiência com o protocolo SNMP, bem como o uso de APIs SNMP para construção de ferramentas de monitorização.

Referências

- [1] <https://tools.ietf.org/html/rfc2790>