TP2 - Monitorização de Ocupação das Partições do SO

João Coelho
(A74859)

11th February 2018

Abstract

Pretende-se criar um programa para monitorização da ocupação das partições de um sistema operativo num qualquer host na rede local. O programa deve disponibilizar uma tabela no monitor da estação de trabalho com a seguinte informação, para cada uma das partições: nome (pequena descrição), tamanho, espaço livre em percentagem.

Para obter a informação, deve ser utilizado o SNMP e as variáveis necessárias de uma MIB adequada.

1 Introdução

Será importante monitorizar as partições do sistema operativo? Bem, a monitorização da utilização do espaço de armazenamento das partições da tua máquina é uma das tarefas mais importantes de um administrador de sistema, uma vez que ajuda a assegurar que existe espaço livre suficiente nestas partições para que o sistema continue a correr de forma segura e eficiente.

Este é o desafio que se pretende resolver com este programa, isto é, criar uma aplicação de monitorização das partições do SO, utilizando um Manager SNMP.

2 Desenvolvimento do programa

2.1 Decisões iniciais

Apresentado o problema, colocavam-se agora questões sobre como resolvê-lo. A primeira decisão passou por escolher as ferramentas (linguagens, frameworks,...) a utilizar, tendo a escolha recaído pelo Perl, para desenvolver a aplicação de backend, onde correria o SNMP, e a framework VueJS 2.0, para o desenvolvimento da interface gráfica.

No caso do Perl, a escolha foi feita com base na existência de um módulo adequado - **[1]** - cuja utilização aparentou uma menor curva de aprendizagem.

Já para a camada de apresentação, escolheu-se esta framework Javascript por uma questão de habituação, que facilitou, assim, o desenvolvimento da interface.

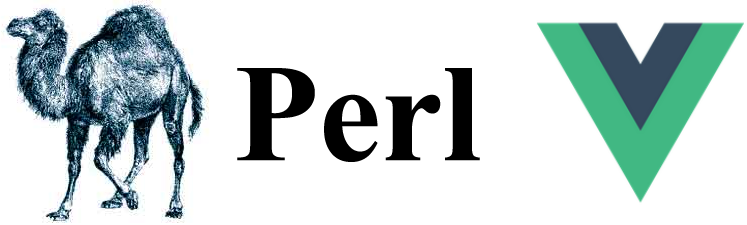

Figure 1: Linguagem e framework utilizadas.

2.2 Implementação

2.2.1 Backend - Perl

Para correr no backend, criou-se um servidor que faz uso essencialmente de 4 módulos Perl:

- SNMP, já abordado anteriormente, que é a base deste programa;
- threads, para permitir o controlo simultâneo das múltiplas partições;
- Net::WebSocket::Server, que possibilita a criação de um WebSocket de interligação entre o servidor e o *client side*;
- JSON, para agrupar e empacotar os dados do servidor, relativos à ocupação das partições, que se pretende enviar para a camada de apresentação, através do WebSocket.

Importados os módulos necessários, a estrutura do programa começa com a abertura da sessão SNMP, que se faz com o comando seguinte, onde importa referir que ARGV[0] representa o endereço IP do host cujas partições serão monitorizadas.

```
my $session : shared = new SNMP::Session(DestHost => $ARGV[0], Version => '2c');
```

O passo seguinte é a descoberta das partições do host, fazendo um pedido getbulk do objeto 'hrPartitionFSIndex' da MIB-II, que, de acordo com a documentação, retorna o índice do File System montado na partição. Para consulta da documentação, a principal referência utilizada foi: [2].

No comando seguinte, importa referir que o 0 e o 1 representam, respetivamente, o número de objetos non-repeaters e max-repeaters da lista que se coloca como terceiro argumento, que neste caso tem apenas um elemento.

Naturalmente, o objeto que se procura não é uma instância, pois o objetivo é descobrir o índice do File System de todas as partições da máquina.

```
($fsIndex) = $session->bulkwalk(0, 1, [['hrPartitionFSIndex']]);
```

A variável fsIndex fica, assim, detentora dos índices dos File Systems das partições, porém, é ainda necessário filtrar esta lista, uma vez que certas partições do sistema não estão associadas a nenhum sistema de ficheiros, retornando -1 quando são alvos de um get do 'hrPartitionFSIndex'. Esta filtragem é feita da seguinte forma:

```
my @ids;
for $i (0..@$fsIndex-1) {
        $id = $$fsIndex[$i]->val;
        if ($id != -1) {
                push @ids, $id;
        }
}
```

Agora sim, @ids possui os índices dos File Systems das partições realmente importantes para monitorização, pelo que podemos passar para a próxima etapa: a obtenção das informações relevantes acerca das partições.

Toda a lógica de obter e enviar a informação das partições para a camada de apresentação, pelo WebSocket, encontra-se numa sub-rotina do programa em Perl, batizada de "getValues" por uma questão lógica, porque, de facto, o que se pretende obter das partições é: o valor de memória total, a memória utilizada, o nome/descrição e, também, as unidades de alocamento de memória.

Esta sub-rotina será chamada na conexão do WebSocket, razão pela qual recebe como argumento três valores:

```
sub getValues {
        my $i = @_[0];
        my $session = @_[1];
        my $conn = @_[2];
```

Aqui, $i representa o índice do File System, $session representa a sessão SNMP e, por último, $conn representa a conexão com o WebSocket.

Antes de enveredar pelo caminho que explica como se conseguiu a interação com o *client side*, continuar-se-á com a descrição do processo de obtenção da informação das partições dentro desta sub-rotina, que se faz num ciclo infinito. Para que se perceba o porquê deste ciclo, adianta-se algo que será visível na próxima fase desta explicação, quando se falar do WebSocket. Como a cada partição do sistema corresponderá um thread, para cada uma será chamada a "getValues", onde este ciclo infinito será responsável por manter a monitorização ativa até que o programa seja interrompido.

```
while (1) {
        {
                lock($session);
                $s = $session->get(['hrFSStorageIndex',$i]);
        }
        $vars = new SNMP::VarList(
                ['hrStorageDescr',$s],
                ['hrStorageSize',$s],
                ['hrStorageUsed',$s],
                ['hrStorageAllocationUnits',$s]);
        {
                lock($session);
                @vals = $session->get($vars);
        }
(...)
```

Analisando a primeira porção do ciclo, verifica-se que, em primeiro lugar, é feito um lock à sessão SNMP, porque as threads das diferentes partições correm concorrentemente, para se obter o índice da storage ('hrStorageEntry') do File System que é monitorizado na thread. Para isto, faz-se um snmpget ao objeto 'hrFSStorageIndex', usando o argumento $i da sub-rotina, que representa o índice do File System como já foi referido.

Com este índice, guardado em $s, é possível obter toda a informação necessária da partição, que será obtida através de snmpgets dos objetos incluídos em $vars, cujas descrições da documentação oficial da MIB dizem, respetivamente:

- 'hrStorageDescr' - Descrição do tipo e instância da storage descrita por esta entrada.
- 'hrStorageSize' - O tamanho da storage representada por esta entrada.
- 'hrStorageUsed' - A quantidade de storage representada por esta entrada que se encontra alocada, em unidades de hrStorageAllocationUnits.
- 'hrStorageAllocationUnits' - Tamanho, em bytes, dos objetos de dados alocados nesta entrada.

Para completar a explicação do bloco de código anterior, resta referir que o segundo lock à sessão é feito para que se possa fazer snmpget dos objetos da MIB abordados anteriormente. Feito o get e colocado o resultado em vals, passa-se à próxima fase: a construção do mensagem JSON a enviar à camada de apresentação.

```
$toGB = $vals[3] / (10 ** 9);
$msg = JSON->new->encode({
        desc => $vals[0],
        size => sprintf("%.2f", $vals[1] * $toGB),
        used => sprintf("%.4f", $vals[2] * $toGB),
        free => sprintf("%.2f", (1 - $vals[2]/$vals[1]) * 100)
});
$conn->send('json', $msg);
(...)
```

A primeira linha deste pequeno bloco de código permite converter para Gigabytes o valor de 'hrStorageAllocationUnits' da storage da partição em análise, algo útil para comparação entre diferentes partições e para que, em termos de apresentação, se tenha uma tabela em que as várias entradas estão na mesma unidade de medida.

A construção da mensagem corresponde, na prática, a criar os campos desc, size, used e free, que representam o nome da partição, o tamanho da sua storage, a quantidade desta já ocupada e a percentagem livre, e atribuir-lhes os valores anteriormente obtidos, podendo ser necessário fazer a conversão para Gigabytes ou, no caso da percentagem livre, mais alguns cálculos ((1 - ocupado/total) * 100). Criada a mensagem, esta é enviada através da conexão do WebSocket, representada por $conn.

Falta, agora, apresentar o último bloco de código da sub-rotina "getValues", que será importante para responder à questão colocada na secção 3. Aqui, mediante a percentagem de espaço livre da partição, o tempo que a thread "dorme", à espera da próxima ronda de monitorização, varia. Deixando a interpretação e o porquê destes tempos para a secção 3, resta descrever brevemente o bloco de código, dizendo que a thread "dorme" 1 segundo se a percentagem de espaço livre ($free) for inferior a 10%, 3 segundos se o espaço livre estiver acima de 10 e abaixo de 30%, 5 segundos se estiver acima de 30 e abaixo de 50% e, por fim, 10 segundos quando o espaço livre é igual ou superior a metade da partição.

```
                $time = ($free < 10) ? 1 : 3;
                $time = ($free >= 10 && $free < 30) ? 3 : 5;
                $time = ($free >= 30 && $free < 50) ? 5 : 10;
                sleep($time);
        }
    }
```

Estando conseguida a informação das partições pretendida, levantava-se então outra dificuldade: como conseguir passar essa informação do backend para o frontend. É nesta fase que entra o seguinte módulo Perl para WebSockets - [3] - que permitiu, precisamente, criar um socket, através do qual ocorre a transmissão dos dados.

```
Net::WebSocket::Server->new(
    listen => 8081,
    on_connect => sub {
        my ($serv, $conn) = @_;
        $conn->on(
            utf8 => sub {
                my ($conn, $msg) = @_;
                my @threads;
                my $pos = 0;
                foreach my $i (@ids) {
                   push @threads, threads->create(getValues($i, $session, $conn));
                }
                foreach my $thr (@threads) {
                   $thr->join();
                }
            },
        );
    },
)->start;
```

Neste último excerto do programa em Perl, pode-se destacar a escolha da porta 8081, onde o WebSocket estará à escuta, e a sub-rotina executada quando se estabelece a conexão, onde essencialmente se cria uma thread por cada partição a monitorizar, executando a sub-rotina "getValues" nessa thread.

2.2.2 Frontend - VueJS

A camada de apresentação é um único componente Vue, que apesar de simples, possui mais linhas de código do que o programa em Perl, sobretudo por concentar o código HTML, Javascript e CSS no mesmo ficheiro.

Deste componente, o mais importante será destacar a conexão ao WebSocket, agora no *client side*, que assenta na seguinte linha e dois listeners, um para abertura do socket e outro para escuta de mensagens que chegam ao socket, vindas do servidor:

```
var ws = new WebSocket("ws://localhost:8081");
```

O listener mais simples simplesmente envia uma mensagem ao servidor, alertando para a abertura da conexão. Já o segundo listener é responsável por identificar se a partição já consta, ou não, da lista existente do lado do cliente, utilizando para tal a função 'findPart'. Em caso afirmativo, uma segunda função é chamada, para atualizar a informação da ocupação da partição ('updatePart'). Caso contrário, a partição é adicionada à lista e a sua informação passa a ser também exibida na tabela.

```
const vm = this;
// Connection opened
ws.addEventListener('open', function (event) {
    ws.send('Starting SNMP Manager application...');
   console.log('Starting SNMP Manager application...');
});
// Listen for messages
ws.addEventListener('message', function (event) {
    var jsonObj = JSON.parse(event.data);
    console.log('Message from server: ', jsonObj);
    if (vm.findPart(jsonObj)){
      vm.updatePart(jsonObj);
    }
    else {
      vm.partitions.push(jsonObj);
    }
});
```

Sendo da opinião que, neste trabalho, a camada de apresentação é importante em termos práticos e não tanto nos detalhes da sua implementação, passa-se agora para a demonstração prática da utilização da aplicação. Deixa-se, desde já, duas notas acerca de duas coisas que não será possível ver no screenshot da próxima secção:

- quando o valor de algum dado da ocupação de uma partição é alterado, ocorre um *flash* na célula correspondente da tabela;
- quando o valor de espaço livre na partição desce dos 50%, a cor com que aparece representado este valor muda de verde para amarelo, passando a vermelho quando o mesmo valor desce dos 10%.

2.3 Utilização

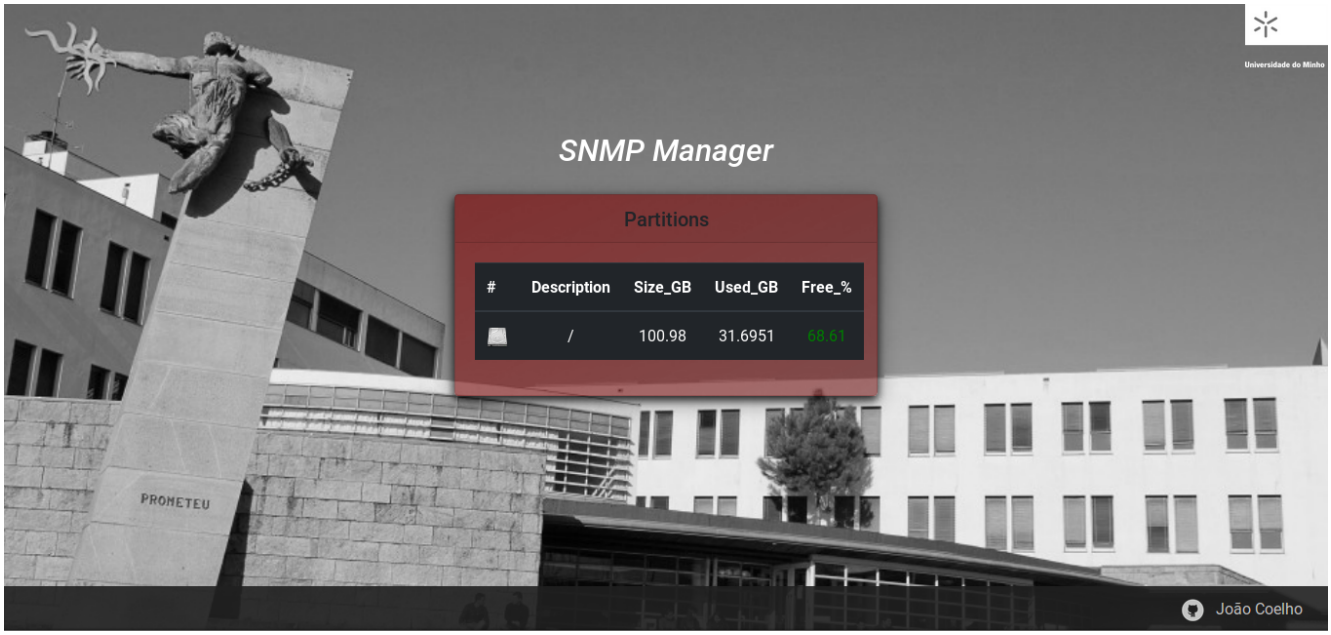

Figure 2: Utilização da aplicação.

3 Questões do enunciado

3.1 Justifique o valor por escolhido para intervalo entre monitorizações consecutivas dos valores das instâncias dos objetos das MIBs que considerou.

Como foi mencionado anteriormente, para responder a esta questão é importante saber o tempo que "dormem" as threads correspondentes às partições em monitorização, porque esse valor representa precisamente o intervalo de tempo entre monitorizações consecutivas. Assim sendo, há necessidade de justificar, não um, mas quatro valores, uma vez que, recorde-se, esse valor está dependente da percentagem de espaço livre na partição, após a última monitorização, podendo variar entre 1 e 10 segundos.

Estes diferentes valores de intervalos entre monitorizações consecutivas são justificáveis pelo facto de, em primeiro lugar, poderem existir múltiplas partições a serem monitorizadas ao mesmo tempo, pelo que o intervalo deve ser adaptado a cada uma, nomeadamente ao seu estado de ocupação, e, em segundo lugar, porque diferentes níveis de ocupação exigem uma monitorização mais ou menos frequente.

Quando se atinge um estado crítico de espaço disponível numa partição (neste caso, inferior a 10%), a monitorização deve ser constante, tendo ainda assim a atenção de não sobrecarregar o sistema, pelo que se escolheu um intervalo de 1 segundo. Em sentido contrário, quando pelo menos metade da partição está livre, a monitorização não necessita de ser tão frequente, motivo pelo qual se faz de 10 em 10 segundos. Para balancear este nível de criticidade de espaço vazio, existem mais dois tempos para o intervalo entre monitorizações consecutivas. São eles os 3 e 5 segundos, o primeiro para estados de ocupação mais críticos (abaixo dos 30% e acima de 10%) e o segundo para um estado entre os 30 e os 50% de ocupação.

4 Conclusão

A aplicação criada permite, através do protocolo SNMP, gerir e monitorizar o estado de ocupação das partições de uma dada máquina, mostrando ao utilizador o espaço de armazenamento da partição, a quantidade deste já usada e a percentagem de espaço livre, além do nome/descrição da partição.

Para uma melhor visualização e controlo, a aplicação apresenta graficamente uma tabela com variações de cor na percentagem de espaço livre, consoante o seu valor, além de exibir um *flash* quando ocorre uma variação numa das células.

Uma alteração futura poderia passar por tornar configuráveis certas características da aplicação, como por exemplo os tempos de intervalo entre monitorizações ou os valores de espaço livre em que ocorre mudança de cor da célula.

References

[1] Snmp - the perl5 'snmp' extension module for the net-snmp package. `http://search.cpan.org/~hardaker/SNMP-5.0404/SNMP.pm`.

[2] Global oid reference database. `oidref.com`.

[3] Net::websocket::server - a straightforward perl websocket server with minimal dependencies. `http://search.cpan.org/~topaz/Net-WebSocket-Server-0.001003/lib/Net/WebSocket/Server.pm`.