



Universidade do Minho
Escola de Engenharia

Gestão e Virtualização de Redes
ENGENHARIA DE TELECOMUNICAÇÕES E INFORMÁTICA
2021/2022

(Docente: Bruno Alexandre Fernandes Dias)

08 de fevereiro de 2021

Trabalho Prático 2

Agente com MIB Virtual e Manager SNMP

Rui Filipe Ribeiro Freitas - pg47639@alunos.uminho.pt

Tiago João Pereira Ferreira - pg47692@alunos.uminho.pt



Índice

Índice de figuras	3
Introdução	4
Fundamentos.....	5
1. Estrutura das ferramentas desenvolvidas.....	5
1.1. MIB	5
1.2. Manager	5
1.3. Agente	6
Desenvolvimento.....	7
1. Construção dos vários componentes	7
1.1. MIB	7
1.2. Manager	8
1.3. Agente	9
2. Manual de utilização	10
Conclusão.....	11



Índice de figuras

Figura 1 - Fluxograma do agente.....	6
Figura 2 - Arquitetura da solução.	7
Figura 3 - MIB.....	7
Figura 4 - Função da comunicação do manager com o agente.....	8
Figura 5 - Função para obtenção da porta.	9
Figura 6 - Excerto de código da interpretação de erros.....	9
Figura 7 - Comandos para iniciar os programas.	10
Figura 8 - Comandos enviados pelo manager e valores recebidos do agente.....	10



Introdução

O presente relatório diz respeito ao trabalho prático número 2 do módulo de Gestão de Redes da unidade curricular de Gestão e Virtualização de Redes. Este trabalho consiste em implementar 2 programas, um para simular um agente com uma MIB virtual e o outro para simular um gestor que pretende obter certas informações da MIB presente no agente.

Este segundo trabalho prático tem como principais objetivos perceber melhor como uma MIB está estruturada e maneiras de podermos obter informações de qualquer agente que contenha uma MIB no mundo real através do SNMP.

Este relatório está dividido em secções que procuram responder ao que é pedido no enunciado como por exemplo a explicação da estrutura das ferramentas desenvolvidas, um pequeno manual de utilização da nossa solução, a estratégia tomada pelo grupo para a construção dos vários componentes como a MIB e os programas do agente e do manager, uma justificação dos objetos e da sintaxe da MIB.



Fundamentos

1. Estrutura das ferramentas desenvolvidas

De modo que houvesse uma melhor organização neste projeto decidimos dividir o trabalho em 3 partes. Primeiro foi necessário elaborar uma MIB virtual que procurasse simular uma MIB real, mas bastante mais simples pois o seu único propósito passaria por ajudar a ter uma melhor perceção sobre as funcionalidades e vantagens da utilização do SNMP. Após ser realizada a MIB foi necessário desenvolver os programas tanto para o manager como para o agente. Estes programas foram realizados com recurso à linguagem C em que os programas comunicam através de *sockets* UDP. Nas subsecções seguintes é possível encontrar uma breve explicação de cada uma destas partes cruciais ao desenvolvimento da nossa solução.

1.1. *MIB*

Relativamente à MIB virtual esta tenta ser parecida ao que uma MIB virtual efetivamente seria, mas de uma forma bastante mais simples tendo em conta os OIDs dos objetos, o tipo de dados de cada objeto, a permissão de cada objeto, ou seja, se o atributo pode ser modificado ou serve apenas de leitura, e também o valor real do objeto. Esta MIB tem também uma linha de cabeçalho que contém alguns conteúdos importantes ao funcionamento do agente, como a porta de onde este deverá estar a ouvir para receber pedidos do manager, a *community string* que serve como palavra de acesso para que o manager possa comunicar com o agente e o OID da MIB sem a especificação dos objetos.

1.2. *Manager*

Quanto ao manager este tem o trabalho de enviar pedidos e receber respostas do agente pelo que no início é aberta a comunicação com o agente à espera de receber um pacote ACK. Mal seja recebido o pacote ACK que indica que um agente se ligou é possível começar a troca de pacotes entre o manager e o agente. Este programa é bastante simples pois o manager apenas é responsável por enviar e receber dados sendo que o trabalho de interpretar e devolver valores está todo no agente.

1.3. Agente

O agente é então o principal componente deste sistema pois é ele que interpreta e procura na MIB as informações requeridas pelo manager. Primeiro o agente começa por realizar a comunicação com o manager através do envio de um ACK de modo que o manager saiba que um agente está conectado e após isso entra num *while* à espera de comandos enviados pelo manager. Estes comandos podem ser de 4 tipos, um *snmpget* em que o agente apenas tem de procurar na MIB o OID do objeto recebido e devolver o valor presente nesse objeto, o *snmpgetnext* que é bastante parecido ao *snmpget* mas este devolve o valor da instância seguinte à do objeto recebido pelo manager, o *snmpgetbulk* que permite que seja recebido mais do que um OID no agente o que faz com que sejam devolvidos vários valores de OIDs no mesmo pacote. Por último, o *snmpset* que permite alterar valores da MIB sendo que é necessária permissão para tal.

Para além de receber e devolver informação de e para o manager o agente é também responsável por identificar problemas que possam ocorrer na receção de dados pedidos. Por exemplo, de modo a aceder ao agente é necessário dizer qual a versão do protocolo SNMP a usar e qual a *community string* sendo que caso algum destes não seja o correto é devolvido ao manager uma mensagem de erro. Outros exemplos de erros que possam ocorrer é caso o manager tente aceder a objetos inexistentes na MIB virtual ou tente alterar valores cuja permissão não o permita. Um último problema que importa salientar é o facto de o agente devolver um erro caso o tipo de dados seja inteiro/*counter* ou similares e o manager tentar alterar para uma *string*.

De seguida é apresentado um fluxograma que clarifica o comportamento do agente tendo em conta os vários pedidos recebidos pelo manager.

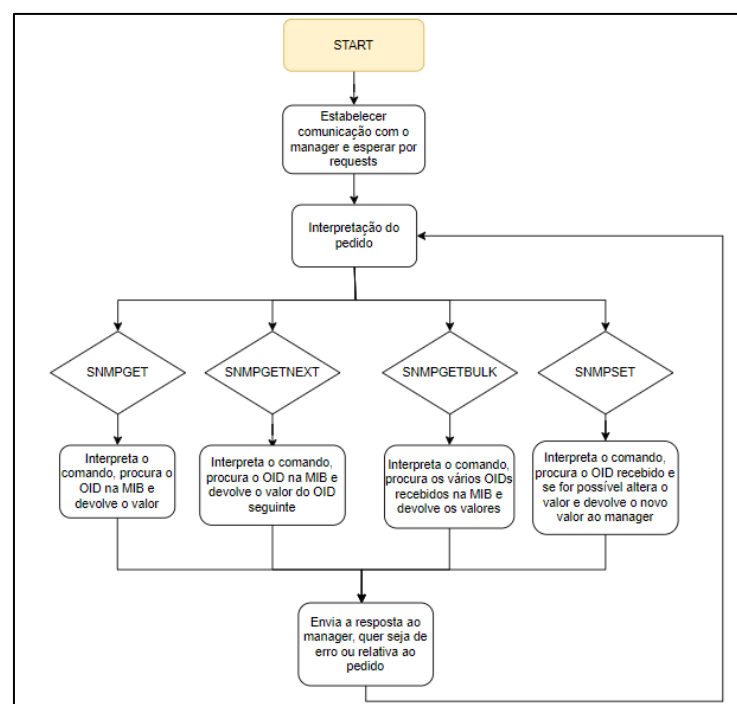


Figura 1 - Fluxograma do agente.

Desenvolvimento

Em relação à solução desenvolvida esta está dividida em 3 partes, a MIB, o agente e o manager. A figura seguinte demonstra o esquema que serviu de base para a solução pretendida.

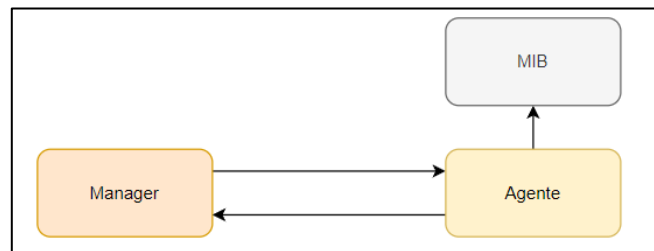


Figura 2 - Arquitetura da solução.

1. Construção dos vários componentes

1.1. MIB

Relativamente ao desenvolvimento da MIB esta foi bastante simples pois apenas foi necessária a realização de um ficheiro texto que seguisse certas regras como a identificação do OID, tipo de dados aceites, permissões para controlo dos dados e o valor. Esta MIB virtual podia ter bastante mais conteúdo, mas achamos ter o suficiente para que fosse possível mostrar a solução que era pretendida. Para além disso estão presentes alguns comentários que identificam melhor cada grupo da MIB. Na figura seguinte é apresentada a MIB com a primeira linha a servir como cabeçalho com informações necessárias ao agente, como a porta e a *community string*. Relativamente aos objetos estes podem tomar vários tipos de dado como Integer, DisplayString, Counter32, IPAddress, entre outros. Também podem tomar valores de leitura apenas ou de leitura e escrita tendo em conta o parâmetro r/rw.

```
≡ MIB.txt
1 8080 public 1.3.6.1.3.1
2 # GRUPO 1 (DATE)
3 .1.1.0 Integer r 2022
4 .1.2.0 Integer r 1
5 .1.3.0 Integer r 27
6 # Grupo 2 (SPECS)
7 .2.1.0 DisplayString rw "MyComputer"
8 .2.2.0 Integer r 4
9 .2.3.0 Counter32 rw 12
10 .2.4.0 IPAddress r 192.168.242.129
11 # Grupo 3 (Random)
12 .3.1.0 Gauge32 rw 123
13 .3.2.0 TimeTicks r 12345
14 .3.3.0 OctetString rw "Hello World"
15
```

Figura 3 - MIB.

1.2. Manager

Relativamente à implementação do manager apenas é de salientar a função de comunicação com o agente demonstrada na figura seguinte. Começa com a criação da *socket* e por dar *bind* da porta para a realização da comunicação. Espera depois pelo ACK enviado pelo agente a dizer que este se encontra ligado. Após isso entra num *loop* onde recebe um input do utilizador com o comando que será enviado para o agente e espera pela resposta. O funcionamento do manager resume-se a isto pois este apenas tem o papel de enviar pedidos e receber respostas.

```
void communication() {
    int sockfd;
    char buffer[MAXLINE];
    struct sockaddr_in servaddr, cliaddr;

    if((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0 ) {
        perror("socket creation failed");
        exit(EXIT_FAILURE);
    }

    memset(&servaddr, 0, sizeof(servaddr));
    memset(&cliaddr, 0, sizeof(cliaddr));

    servaddr.sin_family    = AF_INET;    // IPv4
    servaddr.sin_addr.s_addr = INADDR_ANY;
    servaddr.sin_port = htons(port);

    // Bind the socket with the server address
    if (bind(sockfd, (const struct sockaddr*)&servaddr,
        sizeof(servaddr)) < 0 )
    {
        perror("bind failed");
        exit(EXIT_FAILURE);
    }

    int len, n, x;
    len = sizeof(cliaddr);

    // START COMMUNICATION
    n = recvfrom(sockfd, (char*)buffer, MAXLINE, MSG_WAITALL, (struct sockaddr*)&cliaddr, &len);
    buffer[n] = '\0';

    char command[MAXLINE];
    while (1)
    {
        fflush(stdin);
        // INPUT FROM COMMAND LINE
        printf("Input: ");
        fgets(command, sizeof(command), stdin);
        // SEND COMMAND TO AGENT
        x = sendto(sockfd, command, strlen(command), MSG_CONFIRM, (const struct sockaddr*)&cliaddr, len);
        memset(&command, 0, sizeof(command));

        // WAIT FOR RESPONSE
        n = recvfrom(sockfd, (char*)buffer, MAXLINE, MSG_WAITALL, (struct sockaddr*)&cliaddr, &len);
        buffer[n] = '\0';
        printf("%s\n", buffer);
    }
}
```

Figura 4 - Função da comunicação do manager com o agente.

1.3. Agente

Quanto ao agente este é bastante mais complexo pois trata de toda a interpretação e informação na comunicação. Primeiro é retirado da MIB qual a porta que este se terá de ligar para receber e enviar pedidos. Esta função é demonstrada na figura seguinte. Para além da porta é importante observar que também é retirada do ficheiro a *community string* aceite e o OID da MIB sem a parte dos objetos.

```
void getFromFile(){
    char c[20];
    FILE *fptr;
    if ((fptr = fopen("MIB.txt", "r")) == NULL) {
        printf("Error! File cannot be opened.");
        exit(1);
    }
    fscanf(fptr,"%d %s %s",&port,communityString,mibOID);
    fclose(fptr);
}
```

Figura 5 - Função para obtenção da porta.

Após obter a porta e os restantes atributos é aberta a *socket* como no manager para que a comunicação possa ser efetuada. Após isso entra num *loop* onde vai receber, interpretar e enviar os dados. Relativamente à interpretação que terá sido a parte mais trabalhosa esta tem em conta a sintaxe correta dos comandos recebidos com a função *verifySyntax()* e a interpretação do que se pretende de cada comando com as funções *snmpGet()*, *snmpGetNext()* e *snmpSet()*. Relativamente ao comando *snmpGetBulk*, como este corresponde a um conjunto de *snmpGets* não foi necessário realizar uma função própria. Relativamente à interpretação de comandos errados recebidos pelo agente esta interpretação é feita tendo em conta o valor de erro que a função *verifySyntax()* retorna. Esta secção do código é demonstrada na figura seguinte.

```
while (1)
{
    //printf("WAITING...\n");
    n = recvfrom(sockfd,(char *)line,MAXLINE,MSG_WAITALL,(struct sockaddr*)&servaddr,&len);
    line[strlen(line)] = 0;
    //printf("Line received: %s\n",line);
    error = verifySyntax(line);
    if(error == 1){ // SENT BACK AN ERROR (COMMAND UNKNOWN)
        char errorStr[27] = "Command unknown, try again";
        sendto(sockfd,(const char*)errorStr,strlen(errorStr),MSG_CONFIRM,(const struct sockaddr *) &servaddr, sizeof(servaddr));
    }else if(error == 2){ // SEND BACK AN ERROR (VERSION NOT SUPPORTED)
        char errorStr[33] = "Version not supported, try again";
        sendto(sockfd,(const char*)errorStr,strlen(errorStr),MSG_CONFIRM,(const struct sockaddr *) &servaddr, sizeof(servaddr));
    }else if(error == 3){ // SEND BACK AN ERROR (WRONG COMMUNITY STRING)
        char errorStr[34] = "Wrong community string, try again";
        sendto(sockfd,(const char*)errorStr,strlen(errorStr),MSG_CONFIRM,(const struct sockaddr *) &servaddr, sizeof(servaddr));
    }else if(error == 4){ // SEND BACK AN ERROR (WRONG NUMBER OF ARGUMENTS)
        char errorStr[46] = "Wrong number of arguments, check instructions";
        sendto(sockfd,(const char*)errorStr,strlen(errorStr),MSG_CONFIRM,(const struct sockaddr *) &servaddr, sizeof(servaddr));
    }else if(error == 5){ // SEND BACK AN ERROR (OID NOT COMPATIBLE)
        char errorStr[46] = "OID not compatible, try again";
        sendto(sockfd,(const char*)errorStr,strlen(errorStr),MSG_CONFIRM,(const struct sockaddr *) &servaddr, sizeof(servaddr));
    }
}
```

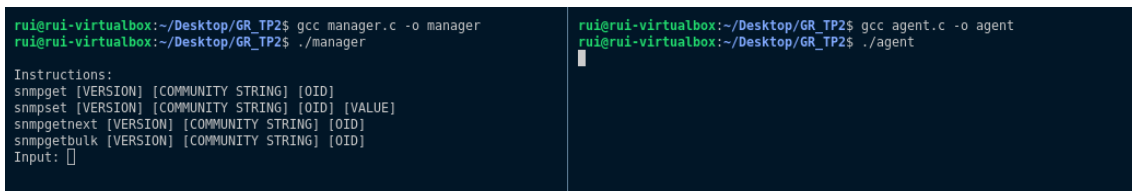
Figura 6 - Excerto de código da interpretação de erros.

2. Manual de utilização

De modo a correr a solução implementada é necessário compilar ambos os programas utilizando os seguintes comandos:

- gcc manager.c -o manager
- gcc agent.c -o agent

Após realizados os comandos anteriores basta correr os programas como demonstrado na figura seguinte.



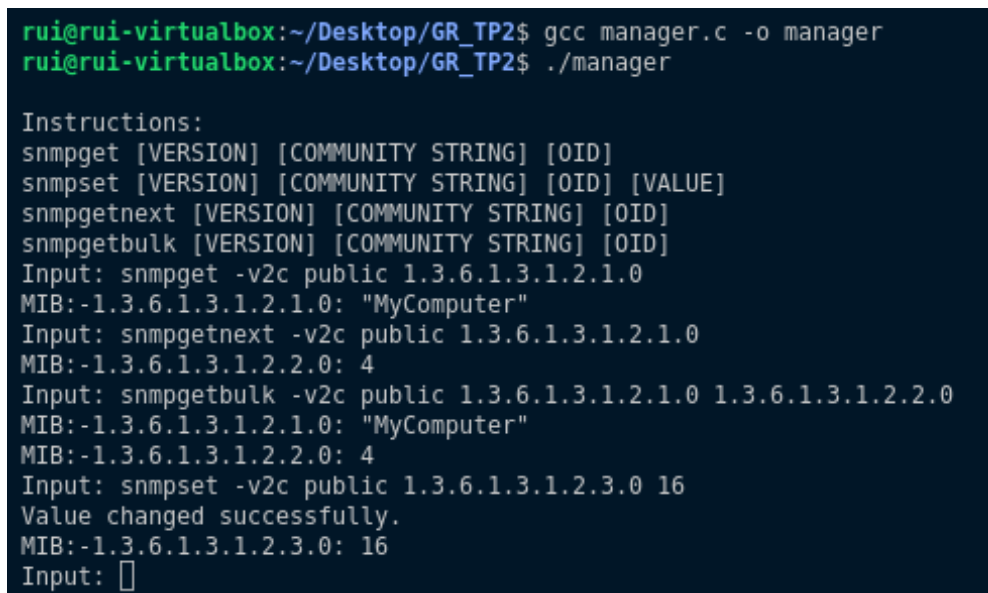
```
ruirui@ruirui-virtualbox:~/Desktop/GR_TP2$ gcc manager.c -o manager
ruirui@ruirui-virtualbox:~/Desktop/GR_TP2$ ./manager

Instructions:
snmpget [VERSION] [COMMUNITY STRING] [OID]
snmpset [VERSION] [COMMUNITY STRING] [OID] [VALUE]
snmpgetnext [VERSION] [COMMUNITY STRING] [OID]
snmpgetbulk [VERSION] [COMMUNITY STRING] [OID]
Input:

ruirui@ruirui-virtualbox:~/Desktop/GR_TP2$ gcc agent.c -o agent
ruirui@ruirui-virtualbox:~/Desktop/GR_TP2$ ./agent
```

Figura 7 - Comandos para iniciar os programas.

Como podemos observar na figura anterior no terminal do manager aparecem algumas instruções de como deve inserir os comandos pretendidos. De seguida é apresentada uma figura com a realização dos 4 tipos de comandos de modo a ser mais prático perceber como são interpretados os comandos na nossa solução.



```
ruirui@ruirui-virtualbox:~/Desktop/GR_TP2$ gcc manager.c -o manager
ruirui@ruirui-virtualbox:~/Desktop/GR_TP2$ ./manager

Instructions:
snmpget [VERSION] [COMMUNITY STRING] [OID]
snmpset [VERSION] [COMMUNITY STRING] [OID] [VALUE]
snmpgetnext [VERSION] [COMMUNITY STRING] [OID]
snmpgetbulk [VERSION] [COMMUNITY STRING] [OID]
Input: snmpget -v2c public 1.3.6.1.3.1.2.1.0
MIB:-1.3.6.1.3.1.2.1.0: "MyComputer"
Input: snmpgetnext -v2c public 1.3.6.1.3.1.2.1.0
MIB:-1.3.6.1.3.1.2.2.0: 4
Input: snmpgetbulk -v2c public 1.3.6.1.3.1.2.1.0 1.3.6.1.3.1.2.2.0
MIB:-1.3.6.1.3.1.2.1.0: "MyComputer"
MIB:-1.3.6.1.3.1.2.2.0: 4
Input: snmpset -v2c public 1.3.6.1.3.1.2.3.0 16
Value changed successfully.
MIB:-1.3.6.1.3.1.2.3.0: 16
Input: 
```

Figura 8 - Comandos enviados pelo manager e valores recebidos do agente.



Conclusão

Com a realização deste trabalho prático adquirimos um conhecimento mais aprofundado sobre o funcionamento das MIBs e sobre a vasta utilidade que o SNMP pode ter. Os objetivos foram todos cumpridos pelo que achamos ter feito um trabalho bem conseguido na realização deste projeto.