



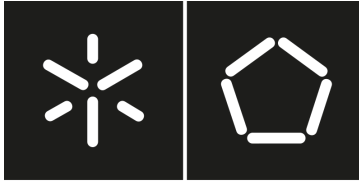
Universidade do Minho
Escola de Engenharia

Catarina Pereira
Inês Neves
Leonardo Martins

SNMPkeys

Catarina da Cunha Malheiro da Silva
Pereira
Inês Cabral Neves
Leonardo Dias Martins

SNMPkeys



Universidade do Minho

Escola de Engenharia

Catarina da Cunha Malheiro da Silva

Pereira

Inês Cabral Neves

Leonardo Dias Martins

SNMPkeys

Relatório de Gestão e Visualização de Redes

Módulo de Gestão de Redes

Mestrado em Engenharia

Telecomunicações e Informática

Trabalho efetuado sob a orientação de:




Professor Doutor António Luís Duarte Costa

**Professor Doutor Bruno Alexandre Fernandes
Dias**

Professor Doutor João Fernandes Pereira

Identificação do Grupo

O grupo é composto pelos seguintes membros, todos pertencentes ao primeiro (1) ano do *Mestrado em Engenharia de Telecomunicações e Informática (METI)*:

Imagem	Nome / Número Mecanográfico / E-mail institucional
	Catarina da Cunha Malheiro da Silva Pereira PG53733 pg537336@alunos.uminho.pt
	Inês Cabral Neves PG53864 pg53864@alunos.uminho.pt
	Leonardo Dias Martins PG53996 pg53996@alunos.uminho.pt

Índice

Identificação do Grupo	ii
Índice de Figuras	iv
Lista de Acrónimos	v
Acrónimos	v
1 Introdução	1
2 Descrição do Problema	2
3 Definição da MIB	4
3.1 Grupo System (“system”)	4
3.2 Grupo Config (“config”)	4
3.3 Grupo Data (“data”)	5
3.4 Entrada “dataTableGeneratedKeysEntry”	5
3.5 Processo de Geração de Chave	6
4 Análise Crítica das Principais Funções	7
5 Casos de Uso	10
6 Conclusão	16

Índice de Figuras

1	Os vários Menus do Projeto.	10
2	Menu de Gerar Chaves do Projeto.	10
3	Data e hora do reinício do sistema.	14
4	As várias variáveis do sistema.	14
5	Visualização das Chaves.	15

Acrónimos

METI	<i>Mestrado em Engenharia de Telecomunicações e Informática</i>
MIB	<i>Management Information Base</i>
OID	<i>Object Identifier</i>
SNMP	<i>Simple Network Management Protocol</i>
TTL	<i>Time-To-Live</i>
UC	<i>Unidade Curricular</i>

1 Introdução

Este relatório faz parte da *Unidade Curricular* (UC) Gestão e Virtualização de Redes, do primeiro (1) semestre do primeiro (1) ano do Mestrado em Engenharia de Telecomunicações e Informática. Este projeto foi desenvolvido como resposta a um problema apresentado pelos docentes para o módulo de Gestão de Redes.

No mundo em constante evolução das tecnologias de comunicação e gestão de redes, a eficiência na supervisão e a monitorização de sistemas torna-se fundamental para garantir a integridade, disponibilidade e desempenho de dispositivos e serviços. Nesse contexto, os protocolos de gestão de rede desempenham um papel crítico, e o *Simple Network Management Protocol* (SNMP) é um dos pilares essenciais.

O projeto tem como objetivo a implementação de um agente e um gestor SNMP, bem como a manipulação de dados Z e a geração de chaves C a partir desses dados. O projeto incorpora uma abordagem prática para ilustrar conceitos-chave de gestão de rede e segurança, explorando o uso do SNMP num ambiente de laboratório.

A presente introdução oferece uma visão geral do contexto do projeto, destacando a importância da gestão de redes e da segurança no âmbito da tecnologia da informação. Além disso, esta fornece uma breve descrição dos principais objetivos do projeto, a sua estrutura e o conteúdo abordado no relatório.

O relatório seguirá com uma discussão detalhada sobre as estratégias adotadas, a definição e manipulação das instâncias dos objetos, a análise das funções e classes implementadas, bem como as conclusões obtidas durante a realização do projeto.

2 Descrição do Problema

O sistema geral envolve uma aplicação Java *multithread* para gerir uma matriz (Z) e executar várias operações relacionadas à sua configuração, criação de chaves e atualizações de matriz. Aqui estão alguns aspectos-chave e pontos de discussão sobre as estratégias escolhidas:

1. Multi-threading:

- A aplicação usa vários *threads* para executar tarefas simultâneas, como atualização da matriz (*UpdateMatrixThread*), monitorização de alterações de configuração do ficheiro F (*ConfigFileThread* - *config.txt*) e o atendimento dos pedidos dos managers (*ClientHandler*).
- O multithreading pretende melhorar o desempenho, permitindo que a aplicação execute diversas tarefas simultaneamente. No entanto, requer uma sincronização cuidadosa para evitar corridas de dados e garantir a segurança do *thread*.

2. Sincronização:

- O código utiliza mecanismos de sincronização (blocos sincronizados) para controlar o acesso a instâncias compartilhados, como a matriz (Z) e parâmetros de configuração.
- A sincronização é crucial em ambientes multithread para evitar condições de corrida e garantir que os threads operem em dados compartilhados de maneira coordenada.

3. Gestão das configurações:

- O *ConfigFileThread* lê os parâmetros de configuração de um arquivo e atualiza diversas variáveis compartilhadas, como Port, K, stringM, T, V, X e a matriz (Z).
- O uso de sincronização e variáveis atômicas (*AtomicInteger*) ajuda a manter a consistência durante as atualizações de configuração.

4. Criação da chave:

- A classe *KeyEngine* gera chaves com base na matriz (Z) e outros parâmetros (K, N).
- O uso da sincronização garante que a geração de chaves seja consistente e evita possíveis problemas ao aceder dados compartilhados.

5. Ficheiro I/O:

- A aplicação lê a configuração do arquivo (*config/config.txt*) e monitoriza as alterações usando o *WatchService*.
- O tratamento adequado de erros (por exemplo, manipulação de *FileNotFoundException*) e o bloqueio de arquivos (*FileLock*) são utilizados.

6. Comunicação UDP:

- A classe *Manager* comunica-se por UDP para executar tarefas como gerar chaves e recuperar informações da *Management Information Base* (MIB).
- O uso de *DatagramSocket* e *DatagramPacket* facilita a comunicação entre o gerenciador e outros componentes.

7. Base de Informações de Gestão (MIB):

- A classe *MIBfunctions* preenche a MIB com vários parâmetros quando esta é inicializada, mas não gere.

- 8. Gestão de tempo:** A classe *TimeManager* gerencia operações relacionadas ao tempo, incluindo cálculo do tempo decorrido, reinicialização do tempo e obtenção de datas e horas salvas.
- 9. Criação de números aleatórios:** Números aleatórios são usados no processo de atualização da matriz (*MatrixFunctions.random*). O uso de `java.util.Random` para geração de números aleatórios é padrão.
- 10. Manipulação de erros:** O código inclui mecanismos básicos de tratamento de erros, como captura de exceções e impressão de mensagens de erro.

Os pontos de discussão importantes para este projeto:

- 1. Desafios de simultaneidade:** Discutir possíveis desafios e soluções relacionadas à simultaneidade, como prevenção de *deadlocks* e sincronização eficiente.
- 2. Dinamismo de configuração:** Explorar as implicações da atualização dinâmica dos parâmetros de configuração durante o tempo de execução e como isso afeta o sistema geral.
- 3. Escalabilidade:** Considerar a escalabilidade do aplicação. Quão bem ele lida com um número crescente de *threads* ou atualizações de configuração.
- 4. Comunicação em Rede:** Discutir a confiabilidade e robustez da estratégia de comunicação UDP. Considerar possíveis problemas como perda de pacotes e falhas de rede.
- 5. Estratégia de Atualização da Matriz:** Discutir a estratégia usada para atualizar a matriz (*Z*) e o seu impacto no comportamento do sistema. Considerar alternativas e compensações.
- 6. Capacidade de manutenção:** Discutir a capacidade de manutenção da base de código, incluindo a clareza do código.
- 7. Estratégias de teste:** Considerar como a aplicação poderia ser testado com eficácia, incluindo testes unitários, testes de integração e testes de um ambiente simultâneo.
- 8. Considerações de segurança:** Explorar possíveis implicações de segurança, especialmente ao lidar com a criação de chaves e comunicação de uma rede.

3 Definição da MIB

Neste capítulo refere-se à definição e explicação detalhada da MIB, incluindo a forma de manipulação dos valores das instâncias dos objetos.

A MIB foi projetada para gerir chaves geradas por um agente SNMP. De seguida é apresentado uma explicação detalhada da estrutura e do propósito de cada parte da MIB, incluindo como os valores das instâncias dos objetos podem ser manipulados:

3.1 Grupo System (“system”)

Este grupo contém informações sobre o sistema e a sua configuração.

1. systemRestartDate

- **Descrição:** Representa a data ($YY \times 104 + MM \times 102 + DD$) quando o agente iniciou uma nova matriz Z.
- **Acesso Máximo:** Somente leitura.

2. systemRestartTime

- **Descrição:** Representa o tempo ($HH \times 104 + MM \times 102 + SS$) quando o agente iniciou uma nova matriz Z.
- **Acesso Máximo:** Somente leitura.

3. systemKeySize

- **Descrição:** Número de bytes (K) de cada chave gerada.
- **Acesso Máximo:** Leitura e escrita.

4. systemIntervalUpdate

- **Descrição:** Número de milissegundos do intervalo de atualização da matriz Z interna.
- **Acesso Máximo:** Leitura e escrita.

5. systemMaxNumberOfKeys

- **Descrição:** Número máximo de chaves geradas que ainda são válidas.
- **Acesso Máximo:** Leitura e escrita.

6. systemKeysTimeToLive

- **Descrição:** Número de segundos de *Time-To-Live* (TTL) das chaves geradas.
- **Acesso Máximo:** Leitura e escrita.

3.2 Grupo Config (“config”)

Este grupo contém informações de configuração.

1. configMasterKey

- **Descrição:** A chave mestra dupla M com pelo menos $K*2$ bytes de tamanho.
- **Acesso Máximo:** Leitura e escrita

2. configFirstCharOfKeysAlphabet

- **Descrição:** O código ASCII do primeiro caractere do alfabeto usado nas chaves (padrão=33).
- **Acesso Máximo:** Leitura e escrita

3. configCardinalityOfKeysAlphabet

- **Descrição:** O número de caracteres (Y) no alfabeto usado nas chaves (padrão=94).
- **Acesso Máximo:** Leitura e escrita

3.3 Grupo Data (“data”)

Este grupo inclui informações sobre as chaves geradas.

1. dataNumberOfValidKeys

- **Descrição:** Número de elementos na tabela ‘dataTableGeneratedKeys’.
- **Acesso Máximo:** Somente leitura.

2. dataTableGeneratedKeys

- **Descrição:** Uma tabela com informações sobre todas as chaves criadas que ainda são válidas.
- **Acesso Máximo:** Não acessível.

3.4 Entrada “dataTableGeneratedKeysEntry”

Representa uma linha da tabela com informações para cada chave.

- **keyId**
 - **Descrição:** A identificação de uma chave gerada.
 - **Acesso Máximo:** Somente de leitura.
- **keyValue**
 - **Descrição:** O valor de uma chave gerada (K bytes/caracteres).
 - **Acesso Máximo:** Somente de leitura.
- **KeyRequester**
 - **Descrição:** A identificação do gerente/cliente que solicitou inicialmente a chave.
 - **Acesso Máximo:** Somente de leitura.
- **keyExpirationDate**
 - **Descrição:** A data ($YY * 104 + MM * 102 + DD$) em que a chave expirará.
 - **Acesso Máximo:** Somente de leitura.
- **keyExpirationTime**
 - **Descrição:** O tempo ($HH * 104 + MM * 102 + SS$) em que a chave expirará.
 - **Acesso Máximo:** Somente de leitura.
- **keyVisibility**
 - **Descrição:**
 0. Valor da chave não é visível;
 1. Valor da chave é visível apenas para o solicitante;
 2. Valor da chave é visível para qualquer pessoa.
 - **Acesso Máximo:** Leitura e escrita.

3.5 Processo de Geração de Chave

Quando um gerente/cliente deseja solicitar a geração de uma chave, ele envia uma solicitação “set()” para gravar 0, 1 ou 2 na instância “keyVisibility.0”. O agente criará uma nova chave (Id=KI) e responderá com o valor da instância “new keyVisibility.KI”. O gerente/agente pode então recuperar qualquer informação necessária do agente, incluindo o valor “keyValue.KI”, usando solicitações “get()”.

A definição da MIB demonstra uma abordagem clara para a gestão de chaves, enfatizando a clareza, eficiência e segurança das operações relacionadas às chaves geradas.

4 Análise Crítica das Principais Funções

Na presente secção apresenta a explicação e análise crítica das principais funções/classes implementadas, os seus principais méritos e a suas limitações mais importantes:

1. KeyEngine Class:

• Principais Méritos:

- A classe “KeyEngine” é responsável pela geração de chaves e gestão do total de chaves geradas.
- A utilização de sincronização (“synchronized”) garante acesso seguro às estruturas de dados compartilhadas.
- O método “generateKey()” implementa a lógica para gerar uma chave com base em operações na matriz Z.
- O método “getTotalKeys()” fornece o total de chaves geradas.

• Limitações:

- Pode haver overhead de desempenho devido à sincronização, especialmente em cenários com muitas threads concorrentes.
- Não há tratamento explícito para situações de erro ou exceções durante a geração de chaves.

2. Manager Class:

• Principais Méritos:

- A classe “Manager” atua como o ponto de entrada do sistema, interagindo com o utilizador e enviando/recebendo dados via UDP.
- Métodos como “setManagerID()” e “getIDPedido()” gerenciam a atribuição de IDs e pedidos, fornecendo uma lógica robusta.
- A implementação de UDP para comunicação entre o gerente e o agente é eficiente para troca de dados.

• Limitações:

- A lógica de espera ativa (“while (lock == null)”) pode ser otimizada para melhorar a eficiência.
- Pode haver questões de concorrência em operações de leitura e gravação nos arquivos de ID e pedidos.

3. MatrixFunctions Class:

• Principais Méritos:

- A classe “MatrixFunctions” contém funções úteis para manipulação de matrizes, como rotação, transposição e operação XOR.
- O uso de “Random” para gerar valores aleatórios é apropriado.
- Métodos como “generateMatrix()” encapsulam a lógica complexa de geração de matrizes.

• Limitações:

- A classe poderia se beneficiar de métodos mais explicativos ou documentação para entender melhor a lógica por trás das operações matriciais.
- Pode haver questões de eficiência dependendo da implementação subjacente de operações matriciais.

4. MIBfunctions Class:

- **Principais Méritos:**

- A classe “MIBfunctions” é responsável por inicializar a MIB com valores padrão.
- Usa um HashMap para armazenar informações da MIB, proporcionando flexibilidade.

- **Limitações:** A dependência de “AtomicInteger” e outros objetos compartilhados não é explicitamente gerenciada nesta classe.

5. TimeManager Class:

- **Principais Méritos:**

- “TimeManager” lida com cálculos relacionados ao tempo, como reinício do sistema e cálculos de data e hora.
- Métodos como “getElapsedTimeInSeconds()” fornecem informações úteis de tempo.

- **Limitações:** A classe poderia incluir mais métodos para manipulação de tempo, como formatação de data e hora para exibição.

6. UpdateMatrixThread Class:

- **Principais Méritos:**

- “UpdateMatrixThread” é uma *thread* dedicada para atualizar dinamicamente a matriz Z com base em intervalos de tempo.
- Usa sincronização para garantir operações seguras em dados compartilhados.

- **Limitações:** A implementação do *loop* infinito pode levar a problemas de fechamento adequado da *thread*.

Ao encerrar este capítulo, é crucial destacar algumas considerações gerais que influenciam a implementação do sistema em análise:

1. Abordagem Clara e Modular:

A implementação adotada reflete uma abordagem clara e modular para lidar com os diversos aspectos do sistema, proporcionando uma estrutura organizada e de fácil compreensão.

2. Escolha do Protocolo UDP:

A opção pela utilização do protocolo UDP para comunicação é uma decisão válida, contudo, é imperativo ponderar sobre as questões relacionadas à confiabilidade e ordem de entrega, as quais exigem uma atenção cuidadosa.

3. Tratamento de Exceções e Erros:

Um ponto crítico a ser observado é que a aplicação não trata completamente cenários de exceção e erros. Recomenda-se, portanto, a implementação de uma gestão mais robusta para fortalecer a estabilidade e confiabilidade do sistema.

4. Incorporação de Logs Detalhados:

A inclusão de logs detalhados pode se mostrar altamente benéfica, permitindo rastrear operações e eventos relevantes durante a execução do sistema. Essa prática contribui para diagnósticos mais eficientes e facilita a identificação de possíveis problemas.

5. Otimização da Lógica de Concorrência:

No contexto de ambientes concorrentes, é sugerido otimizar a lógica de concorrência e sincronização. Essa otimização pode resultar em um melhor desempenho do sistema, proporcionando uma experiência mais fluida em situações de alta concorrência.

5 Casos de Uso

Este capítulo serve para demonstrar como é que o código funciona com todas as possibilidades através dos vários menus, Figura 5. Existe três tipos de menu: o Menu Principal (Figura 1a), o Menu para Gerar as Chaves (Figura 2b) e o Menu sobre a MIB (Figura 1c)

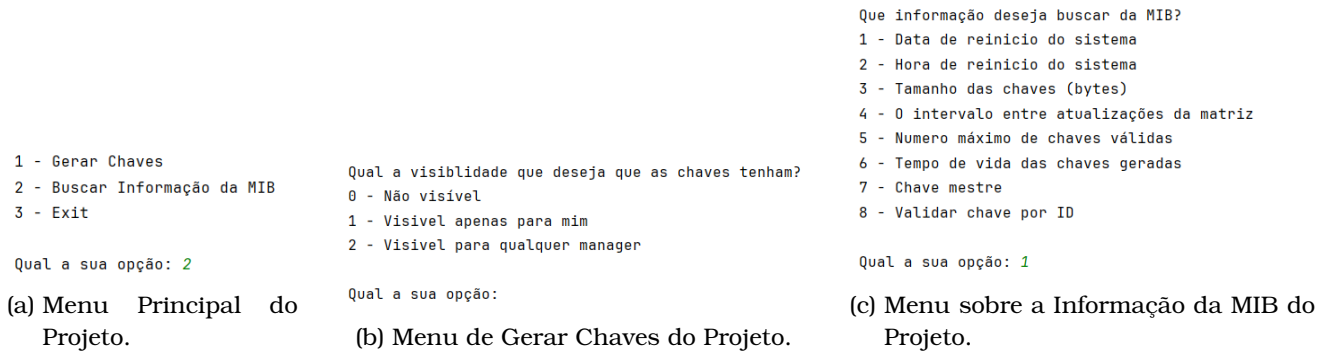


Figura 1: Os vários Menus do Projeto.

Ao fazer *run* primeiramente no Agent e de seguida no Manager, o menu, Figura 1a, é apresentado.

De seguida, ao escolher a opção 1 do Menu irá ser redirecionado para a criação das chaves, Figura 2b, com as três opções:

- 0 - Não visível: tanto o atual manager e outros não conseguem observar, Figura 2a;
- 1 - Visível apenas para mim - o atual manager poder observar mas os outros não podem observar, Figura 2a;
- 2 - Visível para qualquer manager - todos os managers podem observar.

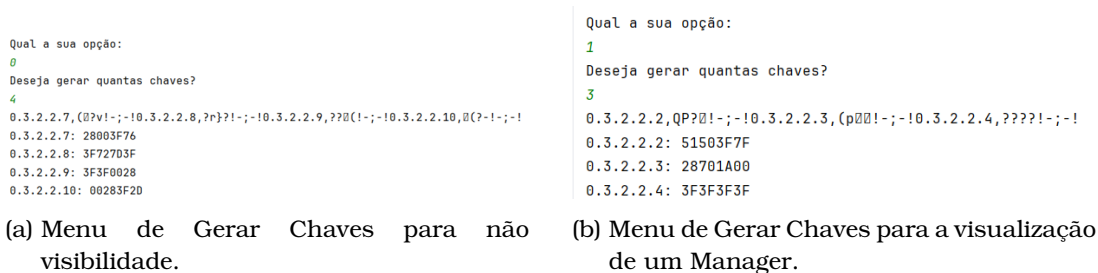


Figura 2: Menu de Gerar Chaves do Projeto.

Ao realizar o pedido de verificação duma chave D enviado pela aplicação, Código 1, é demonstrado dois tipos de mensagens:

- Id de chave inválido - quando o tempo de D é ultrapassado;
- Permissão insuficiente para verificar chave - quando o Manager não tem permissões para a visualização da chave.

O Código 1 está dividida da seguinte forma:

1. Loop Principal (Linhas 75-115):

Um *loop for* que itera sobre um conjunto de dados (chaves). Obtém um identificador chamado *Object Identifier* (OID) a partir de uma *string* dividida por vírgulas. Verifica se o OID começa com a sequência “0.3.2.2.”.

2. Tratamento de OID (Linhas 78-79):
O código divide a *string* do OID usando o ponto como delimitador. A barra invertida (\\) é usada para escapar o ponto, pois o ponto tem um significado especial em expressões regulares.
3. Bloco Sincronizado (Linha 80):
Usa a palavra-chave *synchronized* para garantir que o acesso às estruturas de dados compartilhados seja feito de forma segura em ambientes concorrentes.
4. Acesso a Dados (Linhas 81-84):
Obtém mapas relacionados à gestão de chaves a partir da estrutura MIB.
5. *Switch* de Visibilidade da Chave (Linhas 85-108):
Baseado na visibilidade da chave (*keyVisibility.get(keyID)*), executa diferentes ações.
 - Caso 0: Adiciona uma resposta indicando que o ID da chave é inválido ou que não há permissão para visualizá-la.
 - Caso 1: Verifica se o solicitante da chave é o mesmo que a solicitou originalmente. Se sim, retorna o valor da chave; caso contrário, indica que não há permissão.
 - Caso 2: Retorna o valor da chave diretamente.
6. Construção da Resposta (Linhas 87-112):
Constrói uma *string* de resposta com informações sobre o resultado da verificação da chave, apresentado anteriormente.

Listing 1: Código como é autorizado as chaves com as opções escolhidas

```

75  for(int i = 0; i < Nl; i++){
76      String OID = gets[i].split(",")[0];
77      if (OID.startsWith("0.3.2.2.")){
78          String keyID = OID.split("\\.")[4]; //aqui quebramos a
          ↪ string nos "." mas como "." em regex
79          // significa qualquer caracter entao temos de usar o \\
80          synchronized (MIB) {
81              HashMap<String, Integer> keyIDMap = (HashMap<String,
          ↪ Integer>) MIB.get("0.3.2.1");
82              if(keyIDMap.containsKey(keyID))
83              {
84                  HashMap<String, String> keyValueMap =
          ↪ (HashMap<String, String>) MIB.get("0.3.2.2");
85                  HashMap<String, Integer> keyVisibility =
          ↪ (HashMap<String, Integer>) MIB.get("0.3.2.6");
86                  switch (keyVisibility.get(keyID)){
87                      case 0:
88                          resposta += "0 " + OID + "!--!NULL";
89                          resposta += " 1 " + OID + "--!2"; //erro
          ↪ 2 significa que não tem permissão para
          ↪ ver a chave
90                          break;
91
92                      case 1:
93                          HashMap<String, String> keyRequester =
          ↪ (HashMap<String, String>)
          ↪ MIB.get("0.3.2.3");
94
          ↪ if(keyRequester.get(keyID).equals(camposData[0]))
95                  {
96                      resposta += "1 " + OID + "--!" +
          ↪ keyValueMap.get(keyID);
97                      resposta += " 0";
98                  }
99                  else {
100                      resposta += "0 " + OID + "--!NULL";
101                      resposta += " 1 " + OID + "--!2";
102                      //erro 2 significa que não tem
          ↪ permissão para ver a chave
103                  }
104                  break;
105
106                      case 2:
107                          resposta += OID + "--!" +
          ↪ keyValueMap.get(keyID);
108                          resposta += " 0";
109                          break;
110                  }
111              }
112          ....
113      }
114  }

```

Ao selecionar a opção 2 do Menu 1a pode-se observar o menu das Informações da MIB, Figura 1c.

Dentro deste submenu, a seleção da opção 1 corresponde à visualização da data de reinício do sistema, conforme exemplificado na Figura 3a. Esta informação é obtida por meio da execução do Código 2.

No Código 2, a função `getSavedDate` é responsável por retornar o valor que simboliza a data do reinício do sistema. O método realiza um cálculo, multiplicando o ano por 104, o mês por 102 e somando o dia. Este processo resulta no valor numérico que representa, de maneira específica, a data para o sistema em questão.

Listing 2: Código da data do reinicio do sistema

```
24 public int getSavedDate() {
25     Date date = new java.util.Date(this.startTime);
26     SimpleDateFormat sdfYear = new
27         ↪ java.text.SimpleDateFormat("yy");
27     SimpleDateFormat sdfMonth = new
28         ↪ java.text.SimpleDateFormat("MM");
28     SimpleDateFormat sdfDay = new
29         ↪ java.text.SimpleDateFormat("dd");
29
30     int year = Integer.parseInt(sdfYear.format(date));
31     int month = Integer.parseInt(sdfMonth.format(date));
32     int day = Integer.parseInt(sdfDay.format(date));
33
34     return (year * 104 + month * 102 + day);
35 }
```

Dentro deste submenu, a escolha da opção 2 refere-se à visualização da hora de reinício do sistema, conforme ilustrado na Figura 3b. Este dado específico é obtido através do procedimento delineado no Código 3.

No Código 3, a função `getSavedTime` realiza uma tarefa análoga, porém focada na obtenção da hora de reinício do sistema. Novamente, o cálculo envolve a multiplicação da hora por 104, do minuto por 102 e a adição dos segundos. Este processo resulta em um valor numérico que representa, de maneira específica, a hora para o sistema em questão.

Listing 3: Código da hora do reinício do sistema

```

37 public int getSavedTime() {
38     Date Time = new java.util.Date(this.startTime);
39     SimpleDateFormat sdfHour = new SimpleDateFormat("HH");
40     SimpleDateFormat sdfMinute = new SimpleDateFormat("mm");
41     SimpleDateFormat sdfSecond = new SimpleDateFormat("ss");
42
43     int hour = Integer.parseInt(sdfHour.format(Time));
44     int minute = Integer.parseInt(sdfMinute.format(Time));
45     int second = Integer.parseInt(sdfSecond.format(Time));
46
47     return (hour * 104 + minute * 102 + second);
48 }

```

Qual a sua opção: 1

0.1.1: 3645

(a) Data do reinício do sistema.

Qual a sua opção: 2

0.1.2: 6827

(b) Hora do reinício do sistema.

Figura 3: Data e hora do reinício do sistema.

Neste submenu as opções 3, 4, 5, 6 e 7 indicam várias variáveis respectivamente:

- Tamanho das chaves ($2 \times K$ bytes de comprimento), Figura 4c;
- A matriz Z atualiza a cada intervalo de T milissegundos, Figura 4b;
- O tamanho máximo das chaves, X, Figura 4c;
- A validade das chaves duram V segundos, Figura 4d;
- Chava Mestre, M, Figura 4e.

0.1.3: 4

(a) Tamanho das chaves ($2 \times K$ bytes de comprimento).

Qual a sua opção: 4

0.1.4: 500

(b) A matriz Z atualiza a cada intervalo de T milissegundos.

Qual a sua opção: 5

0.1.5: 10

(c) O tamanho máximo das chaves, X.

0.1.6: 15

(d) A validade das chaves duram V segundos.

Qual a sua opção: 7

0.2.1: teste123

(e) Chava Mestre, M.

Figura 4: As várias variáveis do sistema.

Ao procurar o identificador D da Chave, para conseguir visualizar as chaves. Para além da questão do tempo de visualização (V) também interessa a questão da visibilidade perante os managers.

Os vários tipos de visualização das chaves são os seguintes:

1. Quando se consegue ver as chaves no tempo de visualização e quando os managers têm autorização é representado a mensagem da Figura 5a.

2. Quando as chaves não são visíveis para os managers ou um dos managers só cria chaves para si representa a mensagem da Figura 5b.
3. Quanto o tempo de visualização é ultrapassado representa a mensagem da Figura 5c.

Qual o ID da chave que deseja obter?

4

0.3.2.2.4: 3F3F3F3F

(a) Quando os managers tem autorização suficiente dentro do tempo de visualização.

Qual a sua opção: 8

Qual o ID da chave que deseja obter?

10

0.3.2.2.10: Permissão insuficiente para verificar chave

(b) Quando o manager não tem autorização suficiente.

Qual o ID da chave que deseja obter?

4

0.3.2.2.4: Id de chave inválido

(c) Quando o tempo de visualização foi ultrapassado.

Figura 5: Visualização das Chaves.

6 Conclusão

Em conclusão, este projeto de implementação de um agente e gestor SNMP, juntamente com a manipulação de dados Z e geração de chaves C.

A escolha estratégica de utilizar o protocolo SNMP revelou-se fundamental para alcançar os objetivos propostos. A aplicação demonstrou uma abordagem eficaz na utilização de *multithreading* para melhorar o desempenho, embora tenha sido necessário empregar uma sincronização cuidadosa para garantir a integridade dos dados e evitar condições de corrida.

A discussão detalhada sobre as estratégias adotadas para a gestão de configurações, criação de chaves, comunicação em rede, entre outros aspetos, proporcionou uma compreensão aprofundada das decisões de design e suas implicações no funcionamento do sistema.

A definição da MIB e a explicação detalhada dos grupos e objetos, especialmente no “Grupo System”, contribuíram para a compreensão abrangente da estrutura subjacente que suporta a gestão eficiente das chaves geradas pelo agente SNMP.