Trabalho 2
Redes de Computadores Avançadas

Camila Ilges, Eduardo Pretto, João Pedro Feijó

9 de julho de 2024

Elemento Gerenciado

Para o desenvolvimento deste trabalho, foi simulado um smartwatch, contendo os seguintes atributos:

Status: Indica se o SmartWatch está ligado ou desligado.

Nome: O nome configurável do SmartWatch, que pode ser personalizado pelo usuário.

Daily steps: Contagem de passos dados pelo usuário durante o dia.

Battery level: Percentual atual da bateria do SmartWatch.

Heart rate: Ritmo cardíaco atual medido pelo dispositivo.

Power saving mode: Indica se o modo de economia de energia está ativado.

Sleep cycle: Tabela contendo informações sobre o ciclo de sono do usuário, incluindo fases como sono profundo, leve e REM.

Calories burned: Estimativa de calorias queimadas pelo usuário ao longo do dia.

Distance traveled: Distância percorrida pelo usuário, geralmente medida em quilômetros ou milhas.

Connected to smartphone: Indica se o SmartWatch está conectado a um smartphone.

Notifications: Número de notificações recebidas que ainda não foram vistas pelo usuário.

Step goal: Meta de passos diários definida pelo usuário.

Is charging: Indica se o dispositivo está sendo carregado no momento.

Descrição dos objetos da MIB

Todos os atributos do elemento smartwatch simulado foram traduzidos para objetos na MIB desenvolvida. Além disso, foi criada uma tabela com os dados sobre o sono do usuário.

Tipo DisplayString

Status

- Podendo receber um get
- Acessado pelo OID .1.3.6.1.3.1234.1.1.0

```
status OBJECT-TYPE
    SYNTAX DisplayString
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Current state of the smart watch (on or off)"
    ::= { smartWatch 1 }
```

Name

- Podendo receber um get ou um set
- Acessado pelo OID .1.3.6.1.3.1234.1.2.0

```
name OBJECT-TYPE
    SYNTAX DisplayString
    MAX-ACCESS read-write
    STATUS current
    DESCRIPTION
        "The smart watch name"
    ::= { smartWatch 2 }
```

Tipo Integer32

Daily Steps

- Podendo receber um get
- Acessado pelo OID .1.3.6.1.3.1234.1.3.0

```
dailySteps OBJECT-TYPE
    SYNTAX Integer32
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Total number of steps in the day"
    ::= { smartWatch 3 }
```

Battery Level

- Podendo receber um get
- Acessado pelo OID .1.3.6.1.3.1234.1.4.0

```
batteryLevel OBJECT-TYPE
    SYNTAX Integer32 (0..100)
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Battery percentage of the smart watch"
    ::= { smartWatch 4 }
```

Heart Rate

- Podendo receber um get
- Acessado pelo OID .1.3.6.1.3.1234.1.5.0

```
heartRate OBJECT-TYPE
    SYNTAX Integer32
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Heart rate detected by the smart watch"
    ::= { smartWatch 5 }
```

Calories Burned

- Podendo receber um get
- Acessado pelo OID .1.3.6.1.3.1234.1.7.0

```
caloriesBurned OBJECT-TYPE
    SYNTAX Integer32
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Burned calories during the day"
    ::= { smartWatch 7 }
```

Distance Traveled

- Podendo receber um get
- Acessado pelo OID .1.3.6.1.3.1234.1.8.0

```
distanceTraveled OBJECT-TYPE
    SYNTAX Integer32
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Distance travelled in a day"
    ::= { smartWatch 8 }
```

Notifications

- Podendo receber um get
- Acessado pelo OID .1.3.6.1.3.1234.1.10.0

```
notifications OBJECT-TYPE
    SYNTAX Integer32
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Number of notifications received on the smart watch"
    ::= { smartWatch 10 }
```

Step Goal

- Podendo receber um get ou um set
- Acessado pelo OID .1.3.6.1.3.1234.1.11.0

```
stepGoal OBJECT-TYPE
    SYNTAX Integer32
    MAX-ACCESS read-write
    STATUS current
    DESCRIPTION
        "Steps goal defined for the day"
    ::= { smartWatch 11 }
```

Tipo TruthValue

Power Saving Mode

- Podendo receber um get ou um set
- Acessado pelo OID .1.3.6.1.3.1234.1.6.0

```
powerSavingMode OBJECT-TYPE
    SYNTAX TruthValue
    MAX-ACCESS read-write
    STATUS current
    DESCRIPTION
        "Indicates if power saving mode is on or off"
    ::= { smartWatch 6 }
```

Connected To Smartphone

- Podendo receber um get
- Acessado pelo OID .1.3.6.1.3.1234.1.9.0

```
connectedToSmartphone OBJECT-TYPE
    SYNTAX TruthValue
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Indicates if the smart watch is connected to a phone"
    ::= { smartWatch 9 }
```

Is Charging

- Podendo receber um get
- Acessado pelo OID .1.3.6.1.3.1234.1.12.0

```
isCharging OBJECT-TYPE
    SYNTAX TruthValue
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Indicates if the smart watch is being charged"
    ::= { smartWatch 12 }
```

Tipo SleepCycle

Sleep Cycle Table

- Podendo receber um get
- Acessado pelo OID .1.3.6.1.3.1234.1.13.0

```
sleepCycleTable OBJECT-TYPE
    SYNTAX SEQUENCE OF sleepPhase
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Sleep phases and how many minutes the user got of each one"
    ::= { smartWatch 13 }
```

Sleep Phase

- Podendo receber um get
- Acessado pelo OID .1.3.6.1.3.1234.1.13.1

```
sleepPhase OBJECT-TYPE
    SYNTAX SleepPhase
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Sleep phase"
    INDEX { sleepPhaseId }
    ::= { sleepCycleTable 1 }
```

Sleep Phase ID

- Podendo receber um get
- Acessado pelo OID .1.3.6.1.3.1234.1.13.1.1

```
sleepPhaseId OBJECT-TYPE
    SYNTAX Integer32 { deep_sleep(1), light_sleep(2), rem(3) }
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Sleep phase ID"
    ::= { sleepPhase 1 }
```

Sleep Phase Name

- Podendo receber um get
- Acessado pelo OID .1.3.6.1.3.1234.1.13.1.2

```
sleepPhaseName OBJECT-TYPE
    SYNTAX DisplayString
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Sleep phase name"
    ::= { sleepPhase 2 }
```

Sleeping Time

- Podendo receber um get
- Acessado pelo OID .1.3.6.1.3.1234.1.13.1.3

```
sleepingTime OBJECT-TYPE
    SYNTAX Integer32
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Time spent sleeping"
    ::= { sleepPhase 3 }
```

Detalhes da implementação do agente

A implementação deste agente SNMP para o SmartWatch foi feita em Python, visando gerenciar os parâmetros do dispositivo por meio do protocolo SNMP. O script desenvolvido lê dados de um arquivo JSON que são populados por uma simulação também desenvolvida em Python e salva dados, por meio de um comando set, neste mesmo arquivo.

Carregamento e Salvamento de Dados: O script começa carregando os dados do SmartWatch a partir de um arquivo JSON . A função loadData() é responsável por abrir e ler esse arquivo, convertendo seu conteúdo de JSON para um dicionário Python. A função savedata() é utilizada para salvar qualquer modificação feita nos dados de volta para o arquivo JSON.

Para a manipulação de OIDs através das operações GET, GETNEXT e SET do SNMP foram criadas 4 principais funções, são elas: getOidValue(), handleGet(), handleGetNext(), e handleSet().

getoidvalue() mapeia os OIDs (Identificadores de Objeto) a seus respectivos valores no dicionário de dados, permitindo uma fácil recuperação e atualização de informações. handleget()

Figura 1 – Função load data no código do agente

```
def load_data():
    try:
        with open(json_file_path, 'r') as file:
            return json.load(file)
    except Exception as e:
        print("Failed to load JSON data:", e)
        sys.exit(1)
```

Figura 2 – Função save data no código do agente

```
def save_data():
    try:
        with open(json_file_path, 'w') as file:
            json.dump(smartwatch_data, file, indent=4)
        return "Data saved successfully"
    except Exception as e:
        return f"Failed to save JSON data: {e}"
```

e handlegetnext() são usadas para processar solicitações GET e GETNEXT, respectivamente, buscando os dados solicitados ou o próximo dado disponível na sequência MIB.

Figura 3 – Função getOidValue no código do agente

```
def get_oid_value(oid):
    mappings = {
        ".1.3.6.1.3.1234.1.1.0": ("string", smartwatch_data["status"]),
        ".1.3.6.1.3.1234.1.2.0": ("string", smartwatch_data["name"]),
        ".1.3.6.1.3.1234.1.3.0": ("integer", smartwatch_data["daily_steps"]),
        ".1.3.6.1.3.1234.1.4.0": ("integer", smartwatch_data["battery_level"]),
        ".1.3.6.1.3.1234.1.5.0": ("integer", smartwatch_data["heart_rate"]),
        ".1.3.6.1.3.1234.1.6.0": ("integer", int(smartwatch_data["power_saving_mode"])),
        ".1.3.6.1.3.1234.1.7.0": ("integer", smartwatch_data["calories_burned"]),
        ".1.3.6.1.3.1234.1.8.0": ("integer", smartwatch_data["distance_traveled"]),
        ".1.3.6.1.3.1234.1.9.0": ("integer", int(smartwatch_data["connected_to_smartphone"])),
        ".1.3.6.1.3.1234.1.10.0": ("integer", smartwatch_data["notifications"]),
        ".1.3.6.1.3.1234.1.11.0": ("integer", smartwatch_data["step_goal"]),
        ".1.3.6.1.3.1234.1.12.0": ("integer", int(smartwatch_data["is_charging"])),
        ".1.3.6.1.3.1234.1.13.1": ("integer", smartwatch_data["sleep_cycle"]["deep"]),
        ".1.3.6.1.3.1234.1.13.2": ("integer", smartwatch_data["sleep_cycle"]["light"]),
        ".1.3.6.1.3.1234.1.13.3": ("integer", smartwatch_data["sleep_cycle"]["rem"])
    }
    return mappings if oid is None else mappings.get(oid, (None, None))
```

Figura 4 – Função handleGet no código do agente

```
def handle_get(oid):
    result = get_oid_value(oid)
    if result[0] is not None:
        return f"{oid}\n{result[0]}\n{result[1]}"
    return "NONE"
```

Figura 5 – Função handleGetNext no código do agente

```
def handle_getnext(oid):
    oids = sorted(get_oid_value(None).keys())
    next_oid_index = oids.index(oid) + 1 if oid in oids else 0
    if next_oid_index < len(oids):
        return handle_get(oids[next_oid_index])
    return "NONE"
```

Figura 6 – Função handleSet no código do agente

```python
def handle_set(oid, type, value):
    oid_key = oid.split('.')[-1]
    writable_oids = {
        ".1.3.6.1.3.1234.1.2.0": "name",  # name (read-write)
        ".1.3.6.1.3.1234.1.6.0": "power_saving_mode",  # power_saving_mode (read-write, boolean)
        ".1.3.6.1.3.1234.1.11.0": "step_goal",  # step_goal (read-write)
    }
    if oid in writable_oids:
        if type == "integer":
            smartwatch_data[writable_oids[oid]] = int(value)
        elif type == "string":
            smartwatch_data[writable_oids[oid]] = value
        else:
            return "Unsupported type for SET"
        save_data()
        return f"SET SUCCESS: {oid} set to {value}"
    return "SET FAILURE: OID not writable or does not exist"
```

A função handleset() trata da operação SET, permitindo a modificação de parâmetros configuráveis do SmartWatch, como o nome do dispositivo, o modo de economia de energia e a meta de passos diários. Esta função verifica se o OID é passível de escrita antes de modificar o valor e chama saveData() para persistir essas alterações.

Exemplos de funcionamento do agente

A seguir constam exemplos de execução dos 3 comandos implementados para gerenciamento dos objetos com acesso de somente leitura e leitura/escrita, são eles: snmpget, snmpgetnext e snmpset.

Figura 7 – Exemplo das consultas usando os comandos snmpget e snmpgetnext

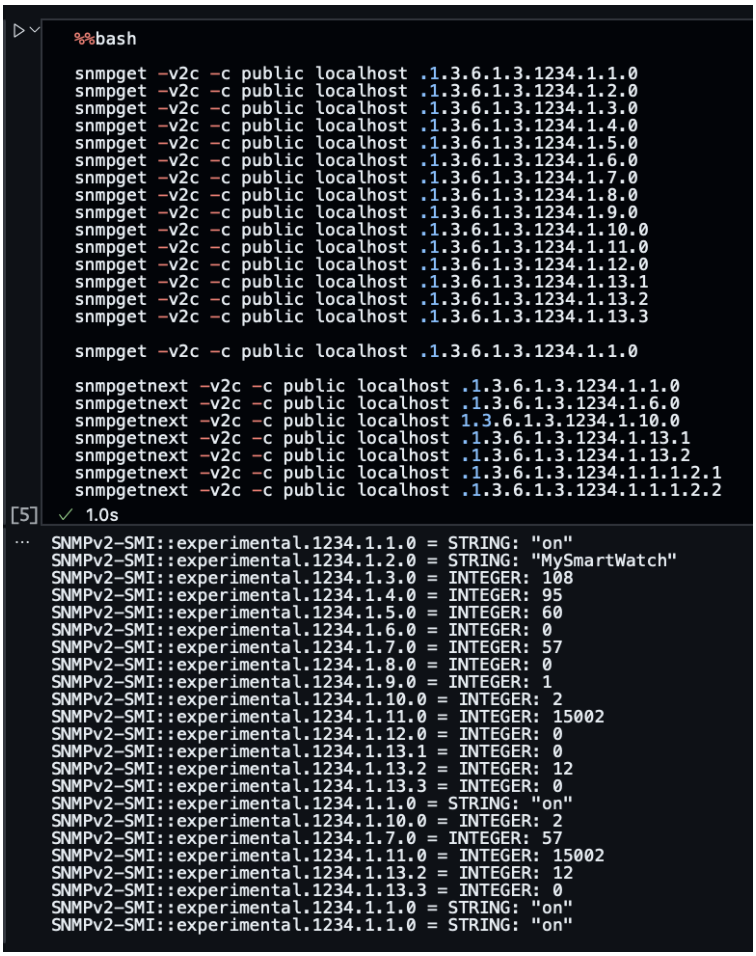

Figura 8 – Exemplo das consultas usando os comandos snmpset sobre um objeto writable e outro objeto read-only e snmpget após para verificar a mudança ou não de valores