REPORT

Internet Protocol Term Project

SNMPv2 Client Implementation

학과	컴퓨터학과
학번	2015410056
이름	김지윤
제출 일자	2018/06/14
담당 교수	김효곤 교수님

Term Project: SNMPv2 Client Implementation

SNMP: Simple Network Management Protocol

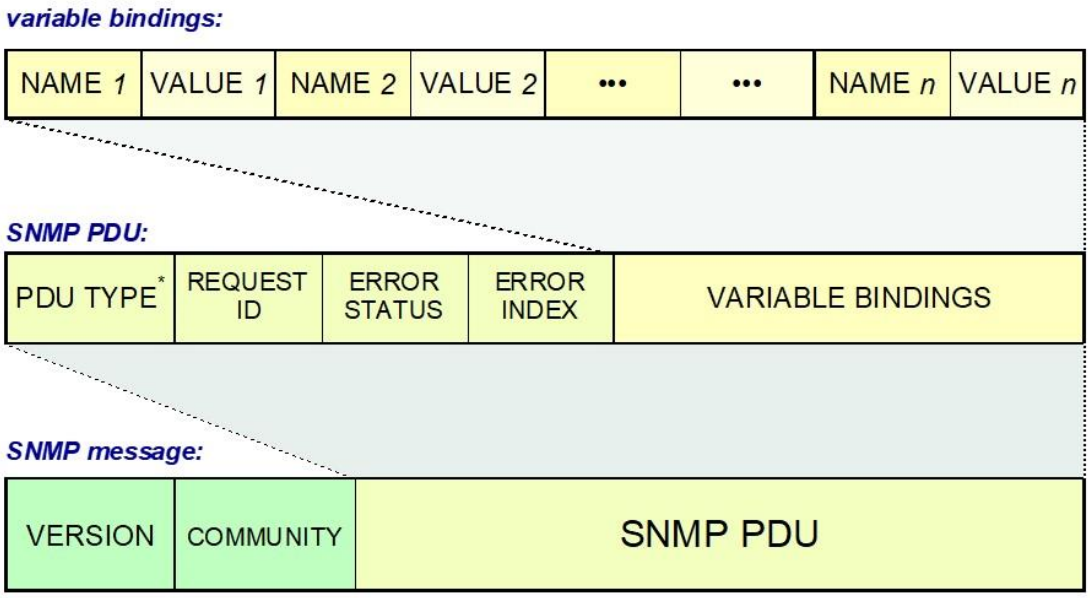
SNMP는 IP 네트워크상의 장치로부터 정보를 수집, 관리, 수정하는 데에 사용되는 프로토콜이다. 이는 SNMP Manager와 SNMP Agent간의 소통을 가능하게 해준다. SNMP Agent (Server)가 가진 특정 정보를 SNMP Manager (Client)는 SNMP를 통해 요청하거나 수정할 수 있다.

■ SNMPv2 Packet Format

1. SNMP Message Structure

SNMP는 SNMP Agent와 Manager사이의 커뮤니케이션을 가능케해주는 프로토콜이며 이 둘은 SNMP Message를 포함한 Packet을 주고받으며 소통한다.

SNMP Message의 구조는 아래 그림과 같다.



SNMP Message가 PDU를 포함하고, PDU는 Variable binding을 포함한다.

a. SNMP Message

- ✓ Version: SNMP의 버전을 나타낸다. SNMPv1(0) SNMPv2(1)
- ✓ Community: 보안 기능을 위한 필드이다. 정보를 받기위한 일종의 패스워드 역할을 한다. 기본 값으로 "public"을 사용하는 경우가 많다.

b. SNMP PDU

- ✓ PDU Type: SNMP 메시지의 정체성을 나타내는 필드이다. GetRequest / Response /

SetRequest등 총 8가지의 type이 존재한다.

- ✓ Request ID: request에 대한 response를 매칭시키기 위한 필드이다. 다수의 프로토콜 패킷이 가지는 Transaction ID 필드와 비슷한 기능을 한다.
- ✓ Error-status: 발생한 에러의 종류를 나타낸다. tooBig(1), noSuchName(2)등 총 5가지의 에러를 나타낼 수 있고, Type이 Request일 경우, 이 필드는 모두 NoError(0)로 채워지게 된다.
- ✓ Error-index: 에러가 발생한 위치를 나타낸다. Error-Status가 noSuchName(2), badValue(3), readOnly(4)일 경우에만 설정된다.

c. Variable Binding

- ✓ Variable Binding: (Variable, Value)

  그림상에는 Name과 Value의 연속이라고 나와있지만 여기서의 Name은 Object ID(OID)를 나타낸다. Request Message일 경우 (OID, null)의 리스트를 전달하고, Response Message일 경우 (OID, 해당 OID가 가리키는 값)이 들어간다.

2. TLV Format

SNMP Message의 대부분의 필드는 TLV Format으로 인코딩 되어있다. TLV format이란 한 필드 (변수)를 Tag, Length, Value의 세가지 요소로 표현하는 것을 말한다.

a. Tag

변수의 type을 1 byte로 나타낸다. SNMP의 문법이라고 할 수 있는 SMI는 아래의 data type을 지원한다.

Data Type	*Tag (Hex)*	*Data Type*	*Tag (Hex)*
INTEGER	**02**	IPAddress	**40**
OCTET STRING	**04**	Counter	**41**
OBJECT IDENTIFIER	**06**	Gauge	**42**
NULL	**05**	TimeTicks	**43**
SEQUENCE, SEQUENCE OF	**30**	Opaque	**44**

b. Length

변수의 길이를 바이트 단위로 나타낸다. 이때 변수의 길이는 TLV포맷 전체가 아니라 value의 길이만을 나타낸다. 길이 값에 따라 표현방식이 다를 수 있는데, 아래 두 가지 경우가 있다.

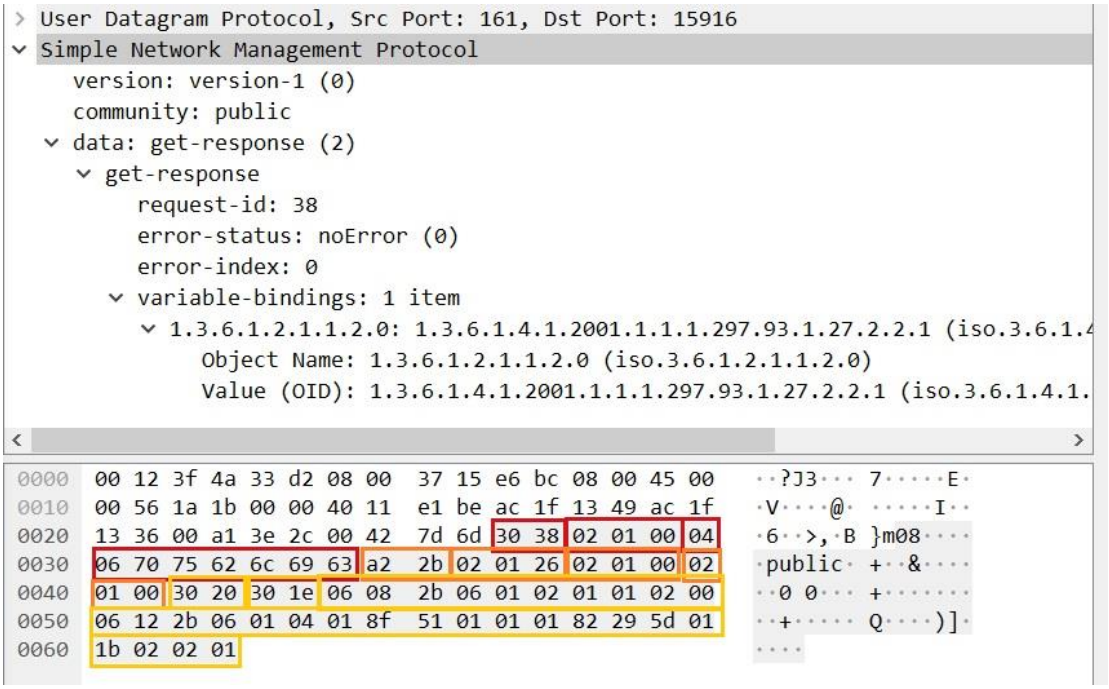
- ✓ 127 byte (0111 1111) 이하일 경우: length 필드의 크기는 1 byte가 된다. 가장 첫 번째 bit는 0, 나머지 7개의 bit가 길이 값을 표현한다.

✓ 128 byte 이상일 경우: length 필드의 크기는 2 byte 이상이 된다. 이때 첫 번째 byte는 실제 length값을 나타내지 않고 이후에 오는 몇 개의 바이트가 length필드인지 알려주는 역할을 한다. 예를 들어 length 필드의 자리에 1000 0011이 채워져 있다면 뒤 이은 3 byte를 length 필드로 사용하겠다는 의미이다. 이 3 byte에 byte단위로 표현된 실제 길이 값이 들어간다.

c. Value

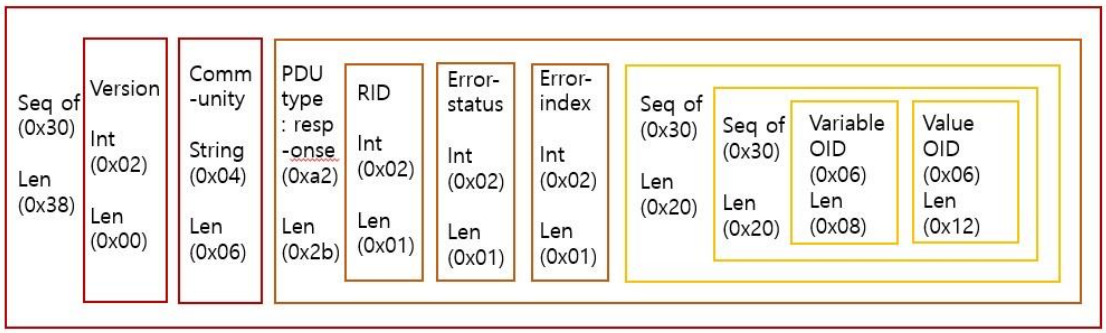
length필드가 끝나면 실제 변수의 value가 인코딩 되어 표현된다.

Wireshark에 캡쳐 된 SNMP 패킷 하나를 분석해보자. 이 SNMP message는 SNMPv1이고 프로젝트에서 구현해야 할 패킷은 SNMPv2이지만 message format에서는 차이를 보이지 않는다.



빨간색으로 표시된 부분을 포함한 전체가 SNMP Message, 주황색 부분부터 뒤 이은 byte들은 SNMP PDU, 노란색 부분이 Variable Binding에 해당한다.

이 패킷 구조를 그림으로 표현하면 아래와 같다. 박스의 색은 위와 매치된다.



- ✓ SNMP message는 여러 type의 TLV 단위들이 묶음이다. → T: sequence of / L: 뒤이어 나타나는 Version + Community + PDU의 길이 / V: SNMP message에 나타나는 모든 요소들

- ✓ Community는 SNMP message의 패스워드 역할을 하는 필드로 데이터 타입은 string 이다. 위 message의 community는 "public"이다. → T: string / L: value의 길이 총 6자리의 octet string이므로 6 byte로 표현 된다. (0x06) / V: 한 byte당 알파벳이 하나씩 ASCII 코드로 인코딩 된다. ('p':0x70 'u':0x75, 'b':0x62, 'l':0x6c, 'i':0x69, 'c':0x63)

- ✓ SNMP message의 header 부분이 끝나면 PDU가 시작된다. 이 또한 TLV format으로서 표현이 되는데, 이때 T는 SMI의 data type이 아닌 PDU type이 된다. 이 패킷은 get-response type이므로 이에 매칭되는 tag a0으로 인코딩 되었다. → T: get-response / L: 뒤이어 나타나는 RID + Error-status + Error-index + Variable Bindings의 길이 / V: PDU에 나타나는 모든 요소들

- ✓ Variable Binding은 Variable(Object ID 또는 Object Name)과 Value pair의 리스트를 나타낸다. 각 variable binding은 sequence of로 묶이고 variable binding들은 또 다시 sequence of 로 묶인다. SNMP message가 단 하나의 variable binding을 포함한다고 해도 형식의 통일성을 위해 sequence of로 한번 더 인코딩한다.

  노란색 부분을 살펴보면 한 쌍의 variable과 value는 sequence of 로 두 번 감싸진 것을 확인할 수 있다.

■ Making SNMPv2 Packet

Term Project는 SNMP Get, SNMP Set, SNMP Walk의 세 가지 기능을 가지는 SNMP Client를 구현하는 것이다. 위 세가지 기능은 모두 request type으로 target OID, Set의 경우 target OID와 더불어 value과 type을 함께 받아 SNMP message에 넣어줘야 한다. 따라서 사용자로부터 입력 받은 input을 SNMP packet으로 인코딩해주는 작업이 필요하다.

1. SNMP Get / SNMP Walk

SNMP Get과 SNMP Walk는 Target OID의 값을 읽는 것이 목적이므로 input으로 OID만이 필요하다. 사용자에게 입력 받은 String형의 OID를 SNMP 인코딩하여 패킷에 넣어주어야 한다

a. OID 변환 함수

String형으로 입력 받은 OID는 패킷에 들어갈 때 노드 값 하나 하나로 인코딩 되어야 하기 때문에 먼저 정수형 배열로 변환해줄 필요가 있다.

```java
//OID 변환 보조함수
public static int[] toIntArr(String oid) {
   String[] str = oid.split("\\.");
   if (str[0].equals("iso")) {str[0] = "1";}
   if (str[1].equals("org")) {str[0] = "3";}
   if (str[2].equals("dod")) {str[0] = "6";}
   int[] result = new int[str.length];
   for (int i=0; i<str.length; i++) {
     result[i] = Integer.parseInt(str[i]);
   }
   return result;
}
```

'.'으로 구분된 Object 노드 값들을 잘라서 int형 배열에 넣고 반환한다.

b. Encoding 과정.

SNMP Client가 보내는 request type의 SNMP 메시지는 Error-Status와 Error-Index필드의 값이 모두 0으로 채워진다.

SNMP Get과 SNMP Walk의 community 필드 값은 "public" 으로 고정 되어있고, Error 관련 필드도 모두 0으로 길이가 고정되어 있으므로 각 필드와 전체 메시지를 계산할 때 고려해야 하는 유일한 변수는 OID뿐이다. 따라서 먼저 OID의 길이를 계산하고 이후에 각 필드 값을 차례로 인코딩한다.

```java
ByteBuffer temp = ByteBuffer.allocate(1024);
BEROutputStream tempBO = new BEROutputStream(temp);
BER.encodeOID(tempBO, BER.OID, this.oid);
BER.encodeHeader(tempBO, BER.NULL, 0);
int varLength = tempBO.getBuffer().position();
// for calculate variable length...
```

임시 outputStream을 만들어서 OID와 Null값을 넣고 position함수로 variable binding의 길이를 구한다.

```java
BER.encodeHeader(berOutput, BER.SEQUENCE, varLength + 26); // sequence
BER.encodeInteger(berOutput, BER.INTEGER, 1); // version
BER.encodeString(berOutput, BER.OCTETSTRING, "public".getBytes()); // commnunity
BER.encodeHeader(berOutput, (byte)getRequest, varLength + 13); // PDU header
BER.encodeInteger(berOutput, BER.INTEGER, requestID); // Request ID
BER.encodeInteger(berOutput, BER.INTEGER, 0); // error status
BER.encodeInteger(berOutput, BER.INTEGER, 0); // error index
BER.encodeHeader(berOutput, BER.SEQUENCE, varLength+2); // sequence of
BER.encodeHeader(berOutput, BER.SEQUENCE, varLength); // sequence
BER.encodeOID(berOutput, BER.OID, this.oid); // oid
BER.encodeHeader(berOutput, BER.NULL, 0); // value
```

✓ 인코딩에는 BER class의 encode[Type] 함수를 사용한다.

✓ Request message므로 variable binding의 value 필드에는 null을 넣어준다.

- ✓ L값이 varLength인 sequence of 타입으로 Variable binding을 한번 감싸고 L값에 2를 더해 (안쪽 sequence of의 T + L 1 byte씩) 한번 더 감싸준다.
- ✓ PDU header의 L값은 varLength + 13이 되는데, Variable binding의 L(varLength + 2) + Variable binding의 T와 L(2) + error-status(3) + error-index(3) + requestID(3)을 모두 더한 값이다.
- ✓ 전체 message의 길이 값은 PDU에서 13만큼이 더 붙게 되는데, community 필드(8) + version 필드 (3) PDU의 T+L(2)가 더해진 값이다.
- ✓ PDU type은 각각 SNMP Get은 Get Request, SNMP Walk은 Get Next Request이다.

2. SNMP Set

전체적인 인코딩 방식은 위와 동일하지만 input으로 OID와 더불어 type과 value가 함께 입력되고 패킷에 포함시켜야 하기 때문에 varLength값을 계산에 추가적인 과정이 필요하고, community 값이 SNMP Get / SNMP Walk와 다르게 "write" 5자리이기 때문에 전체 message의 길이 값도 varLength + 25로 기본 길이 값이 1 적게 나타난다.

```
BER.encodeOID(tempBO1, BER.OID, this.oid);
if (this.type == INTEGER) BER.encodeInteger(tempBO1, BER.INTEGER, Integer.parseInt(this.value));
else if (this.type == STRING) BER.encodeString(tempBO1, BER.OCTETSTRING, this.value.getBytes());
else if (this.type == NULL) BER.encodeHeader(tempBO1, BER.NULL, 0);
else if (this.type == OID) BER.encodeOID(tempBO1, BER.OID, toIntArr(this.value));
int varLength = tempBO1.getBuffer().position();
// for calculate variable length...

BER.encodeHeader(berOutput1, BER.SEQUENCE, varLength + 25); // sequence
BER.encodeInteger(berOutput1, BER.INTEGER, 1); // version
BER.encodeString(berOutput1, BER.OCTETSTRING, "write".getBytes()); // commnunity
```

■ Description of Application Call Flow

1. SNMPClient Thread

SNMP Client 프로그램의 세 가지 기능에 구분을 두지 않고 모두 하나의 Thread **snmpClient**에서 처리하였다. snmpClient는 run, encodeData, decodeData 세 개의 메소드를 가지며 각 메소드 내부에선 선택한 기능(SNMPGet / SNMPSet / SNMPWalk)에 따라 경우를 나누어 처리하였다.

2. Encoding part (encodeData)

[Making SNMPv2 Packet]부분에서 설명한 것과 같이 인코딩하여 output stream을 만든 후 byte buffer를 리턴 한다.

3. Decoding part (decodeData)

보낸 request 패킷의 결과로 돌아오는 response 패킷을 디코딩한다. SNMP header부터 version, community 차례로 디코딩하고 마지막에 나오는 variable binding의 OID와 value를 출력한다. 이 때 Value의 T값에 따라 V를 다르게 출력해야 한다.

```
BER.decodeHeader(berInput, mutData); // decode snmp header
BER.decodeInteger(berInput, mutData); // decode snmp ver
BER.decodeString(berInput, mutData); // decode snmp community
BER.decodeHeader(berInput, mutData); // decode pdu header
BER.decodeInteger(berInput, mutData); // decode requestID
BER.decodeInteger(berInput, mutData); // decode error status
BER.decodeInteger(berInput, mutData); // decode error index
BER.decodeHeader(berInput, mutData); // decode sequence of
BER.decodeHeader(berInput, mutData); // decode sequence
oid = Arrays.toString(BER.decodeOID(berInput, mutData)); // decode oid
```

```
tag = berInput.read();
berInput.getBuffer().position(berInput.getBuffer().position() - 1);
switch (tag) {
case BER.INTEGER: {
   return makeResultOID(oid) + " =" + " INTEGER " + Integer.toString(BER.decodeInteger(berInput, mutData));
}
case BER.OCTETSTRING: {
   String temp = new String(BER.decodeString(berInput, mutData));
   return makeResultOID(oid) + " =" + " STRING " + temp;
}
case BER.OID: {
   return makeResultOID(oid) + " =" + " OID " + makeResultOID(Arrays.toString(BER.decodeOID(berInput, mutData)));
}
case BER.NOSUCHOBJECT: {
   return makeResultOID(oid) + " =" + " No such object";
}
case BER.TIMETICKS: {
   Integer timeticks = BER.decodeInteger(berInput, mutData);
   return makeResultOID(oid) + " =" + " Timeticks " + timeticks.toString();
}
default: {
   return makeResultOID(oid) + " =" + " Type undefined";
}
```

tag값에 따라 value를 디코딩한다. 이때 OID에 대해서는 makeResultOID라는 함수를 호출하고 있는데, 이는 정수 배열 형태를 띄는 string을 Object name 형식에 맞게 변환해주는 함수이다.

```
//OID conversion (arr shaped) string to string
public static String makeResultOID(String oid) {
  String result = oid.substring(1, oid.length() - 1);
  result = result.replace(", ", ".");
  result = "iso" + result.substring(1);
  return result;
}
```

(ex [1, 3, 6, 1, 2, 1, 1] → iso.3.6.1.2.1.1)

디코딩의 경우도 마찬가지로 SNMPWalk에 대해서만 반복 수행한다.

4. Run (run)

Host address와 port를 과제에서 주어진 대로 설정하여 encodeData() 메소드를 통해 인코딩된 데이터를 패킷에 담아 보내고, 그 결과로 돌아오는 response packet을 받아, decodeData() 메소드를 통해 디코딩 한 후 출력하는 기능을 한다.

SNMPGet / SNMPSet / SNMPWalk 모두 큰 틀은 같지만, SNMPWalk의 경우 subtree가 끝날 때 까지 위의 기능을 반복 수행해야 하기 때문에 while문이 추가된다.

Subtree에서 아직 순회할 노드가 더 남았는지 여부는 아래의 함수로 판별한다.

```
   //SNMP Walk -> subtree
   public static boolean subtree(int[] init, int[] respOID){
      for (int i=0; i<init.length; i++) {
         if (init[i] != respOID[i]) return false;
      }
      return true;
   }
}
```

Response로 돌아온 OID가 사용자가 input으로 넣어준 OID를 prefix로 가지는지 체크하고 그렇지 않다면 더 이상 순회할 자식 노드가 없는 것으로 판단한다.

5. Main function

Android Application으로 구현하는 것을 완성하지 못해, 콘솔에서 조작할 수 있도록 UI를 구성하였다.

```
while(state)
{System.out.println("Select function with number");
 System.out.println("1. snmpGet");
System.out.println("2. snmpSet");
System.out.println("3. snmpWalk");
System.out.println("0. Terminate the program");
```

int 입력으로 기능을 고를 수 있다. 0번을 입력할 때 까지 계속해서 기능선택을 반복할 수 있으며 입력으로 0번이 들어오면 while문을 빠져나가, log파일을 생성하고 프로그램을 마친다.

```
if (sel == 1) {
   // snmp get
   snmpClient snmp = new snmpClient(socket, oid);
   snmp.run();
   System.out.println("");
}
```

SNMP Get을 선택했을 경우이다. 위에서 입력 받은 OID를 snmpClient thread에 넘겨주고 run을 수행한다.

```
else if (sel == 2) {
    // snmp set
    System.out.print("Value: ");
    value = sc3.nextLine();
    System.out.print("Type of Value: ");
    type = sc4.nextLine();

    if(value.isEmpty() || type.isEmpty()){
      System.out.print("Error: Wrong input");
      break;
    }
    snmpClient snmp = new snmpClient(socket, oid, value, type);
    snmp.run();
    System.out.println("");
}
```

SNMP Set을 선택한 경우이다. Type과 Value를 위한 추가적인 입력을 받고 있고, OID와 함께 snmpClient thread로 넘겨주고 있다.

```
else if (sel == 3) {
    // snmp walk
    log = "Input OID: "+ oid+ "\r\n";
    snmpClient snmp = new snmpClient(socket, oid);
    snmp.run();
    System.out.println("End of SNMP Walk\n");
    logResult = logResult.concat(log+"\r\n");
}
```

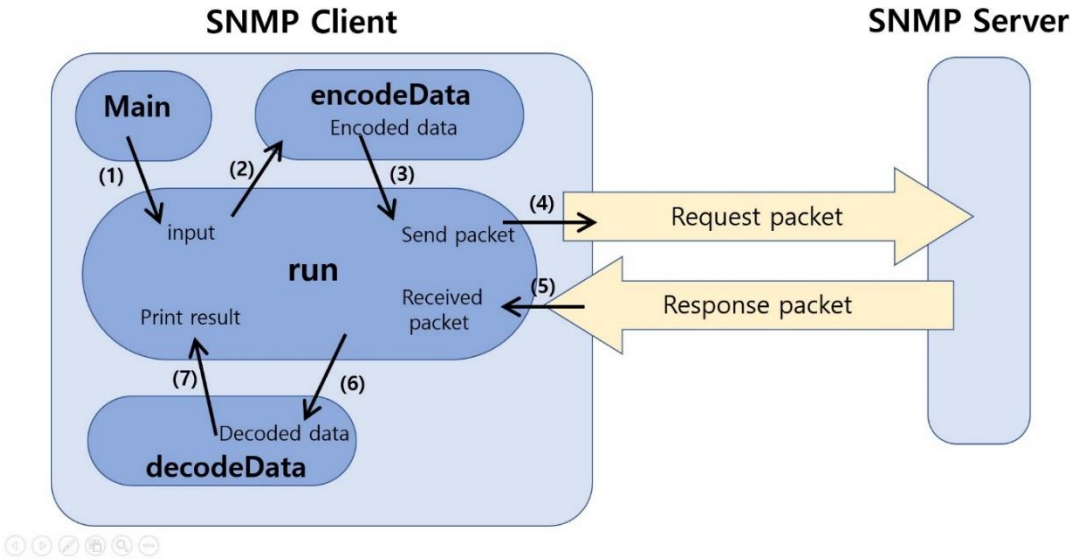
SNMP Walk을 선택한 경우이다. SNMP Get과 비슷하지만 다른 점은 log파일을 위한 부분이다. log파일에 넣을 String인 log에 입력한 OID를 넣어주고, run() 메소드를 돌리며 결과를 출력할 때마다 축적했던 log데이터를 logResult에 합쳐준다.

```
public static void makingLog(String log) throws Exception {
  FileWriter fw = new FileWriter("SNMPwalkLog.txt");
  String text = log;
  fw.write(log);
  fw.close();
}
```

logResult는 로그파일 생성 함수에 넘겨져서 SNMPwalkLog.txt파일에 저장된다.

6. 실행흐름

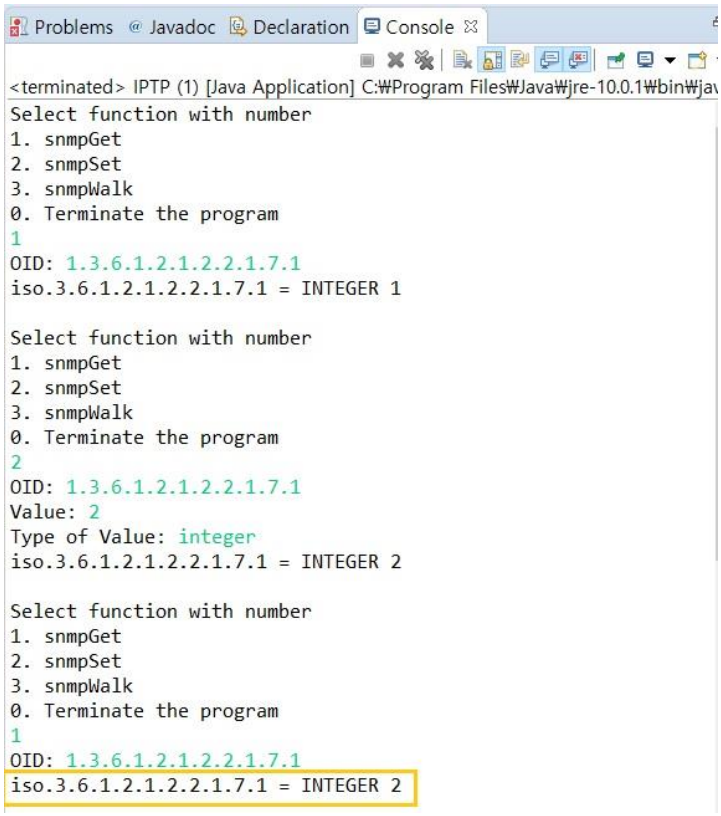
개략적인 실행 흐름은 아래 그림과 같다. SNMP Walk의 경우 Get next를 수행하면서 4~7번 과정을 반복하게 된다.



■ Image Capture of Testing functions

안드로이드 앱 프로그래밍을 해본 경험이 없어 JAVA로 기능 코드만 구현하였다.

SNMP Get (1.3.6.1.2.1.2.2.1.7.1) → SNMP Set (1.3.6.1.2.1.2.2.1.7.1, integer, 2) → SNMP Get (1.3.6.1.2.1.2.2.1.7.1) → SNMP Walk (1.3.6.1.2.1.1) → SNMP Walk (1.3.6.1.2.1.1.9) 순서대로 실행하고 종료한 모습이다.

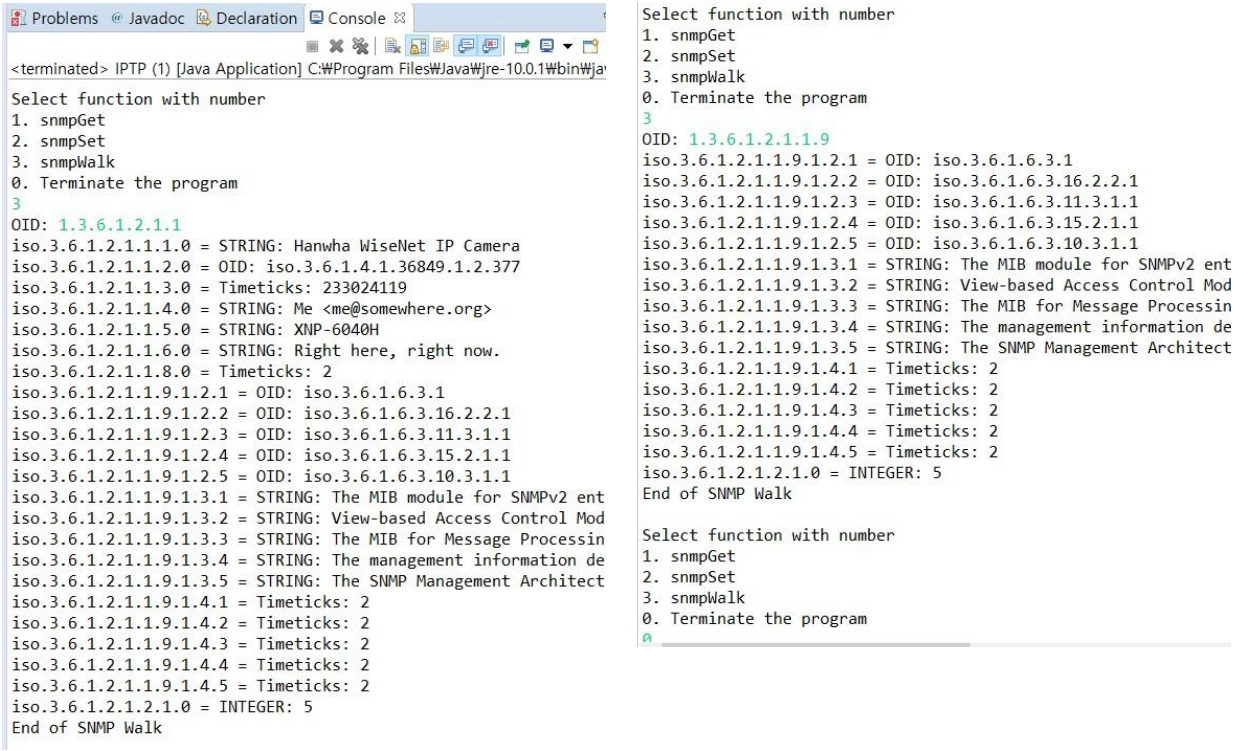


1번을 선택하고

OID 1.3.6.1.2.1.2.2.1.7.1을 입력하여 해당 값을 확인하였다. → INTEGER 1

2번을 선택하고 이전과 같은 OID값을 Target ID로 하여 INTEGER 2로 Set request하였다.

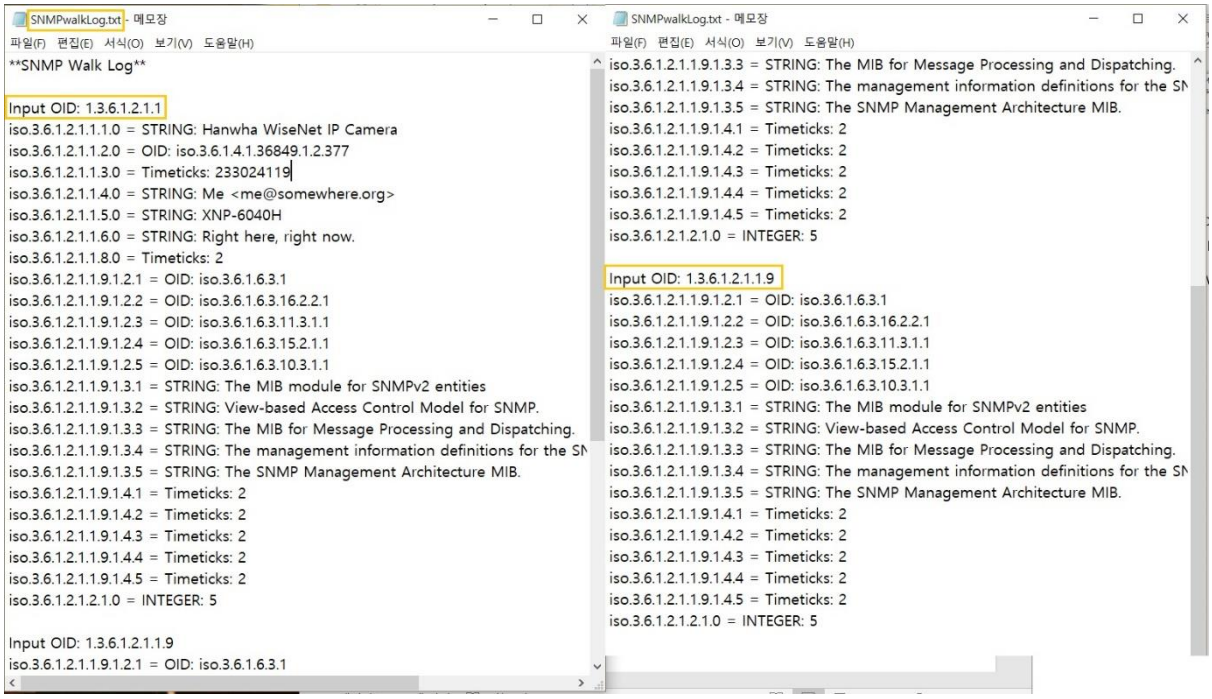
다시 1번을 선택하고 위의 OID를 입력해 확인해보면 값이 INTEGER 2로 바뀌어 있는 것을 확인할 수 있다.



왼쪽은 OID 1.3.6.1.2.1.1을 target으로 하여 SNMP Walk를 실행한 모습이고 오른쪽은 OID 1.3.6.1.2.1.1.9를 target으로 하여 SNMP Walk를 실행한 후 0을 입력해 종료한 모습이다.

실행이 종료되기 직전 프로그램은 SNMP Walk에 대한 log파일을 생성한다.

생성된 SNMPwalkLog.txt 파일을 열어보면 위에서 실행한 두 차례의 SNMP Walk에 대한 결과가 차례대로 기록되어있는 것을 확인할 수 있다.



■ **Reference** (그림 출처 및 코드 참고)

- ✓ https://widen.korea.ac.kr
- ✓ http://www.net-snmp.org/docs/man/
- ✓ http://sanghoon2.tistory.com/269
- ✓ https://blog.jayway.com/2010/05/21/introduction-to-snmp4j/