Lab 02: Đi sâu vào Tic-Toc Tutorial

1. Các file NED dùng để làm gì ?

OMNeT ++ sử dụng các tệp NED để định nghĩa các thành phần và để lắp ráp chúng thành các đơn vị lớn hơn như mạng

2. File omnetpp.ini dùng để làm gì?

Tệp *omnetpp.ini* nói cho trình mô phỏng biết chúng ta muốn mô phỏng mạng nào (vì các tệp NED có thể chứa nhiều mạng), ta có thể truyền tham số cho model, chỉ định rõ ràng các seed cho trình tạo số ngẫu nhiên, v.v.

3. Ý nghĩa của đoạn mã nguồn dưới đây:

"tic.out --> { delay = 100ms; } --> toc.in;"

Đoạn mã trên có nghĩa là:

Cổng *out* của mô đun *tic* được kết nối với cổng *in* của mô đun *toc* thông qua kênh truyền có độ trễ 100ms

4. Hãy giải thích đoạn mã nguồn sau:

```
if (strcmp("tic", getName()) == 0) {
        cMessage *msg = new cMessage("tictocMsg");
        send(msg, "out");

    }
```

Đoạn mã trên kiểm tra xem mô đun hiện tại có phải có tên là "tic" hay không. Nếu đúng thì sẽ tạo ra một thông điệp có nội dung "tictocMsg" và gửi đến cổng out của nó.

5. Làm thế nào mà các module *tic* và *toc* tách biệt được với nhau?

tic và toc được tách biệt với nhau dựa trên sự định nghĩa chúng ở trong tệp NED (tên là gì, thuộc kiểu nào, tham số bằng bao nhiêu,...)

ví dụ:

```
tic: Txc4 {
    parameters:
        sendMsgOnInit = true;
        @display("i=,cyan");
}
toc: Txc4 {
    parameters:
        sendMsgOnInit = false;
        @display("i=,gold");
}
```

mô đun thứ nhất tên là "tic", thuộc kiểu "sample module Txc4", có tham số *sendMsgOnInit*(gửi tin nhắn khi khởi tạo) bằng true và hiển thị ở màu lục lam trên trình mô phỏng.

mô đun thứ hai tên là "toc", thuộc kiểu Txc4, tham số *sendMsgOnInit* = false, hiển thị ở màu vàng trên trình mô phỏng.

6. Xác định vai trò của các biến *counter* và *limit*, giải thích ý nghĩa của chúng.

Tham số *limit* được định nghĩa trong tệp NED, dùng để lưu giới hạn số lần nhận-gửi thông điệp của một mô đun. Biến *counter* được khai báo trong class Txc, có giá trị khi khởi tạo bằng giá trị của tham số *limit*, thông qua biểu thức counter = par("limit") . Mỗi lần mô đun tic hoặc toc nhận được một thông điệp, biến *counter* sẽ bị giảm đi một giá trị, đến khi nó bằng 0 thì sẽ dừng quá trình gửi và nhận giữa tic-toc.

Tham số *limit* có thể được gán giá trị trực tiếp khi khai báo trong tệp NED, hoặc có thể được gán trong khối *parameters* của mô đun con kế thừa, hoặc có thể được gán thông qua tệp INI, hoặc cũng có thể được nhập vào từ giao diện mô phỏng. Chương trình sẽ nhận *limit* như một tham số đầu vào.

Biến *counter* là một biến trạng thái (state variable), dùng để lưu các trạng thái của mô đun ở các thời điểm xác định. Các biến trạng thái có thể được theo dõi ở trong cửa sổ *inspector*, nếu ta thêm câu lệnh WATCH([tên biến]) trong hàm init.

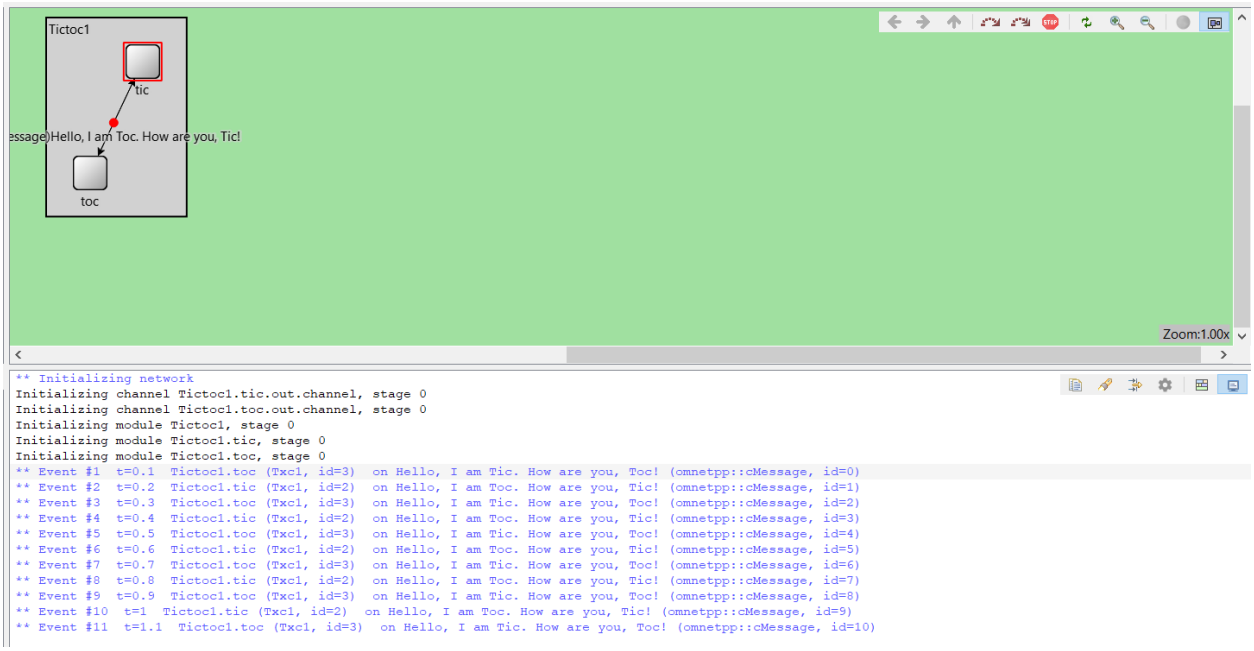
7. Chỉnh sửa mã nguồn sao cho thông điệp gửi từ *tic* đến *toc* là "Hello, I am Tic. How are you, Toc!" và ngược lại.

sửa hai hàm trong tệp txc1.cc:

```
void Txc1::initialize()
{
    if (strcmp("tic", getName()) == 0) {
        cMessage *msg = new cMessage("Hello, I am Tic. How are you, Toc!");
        send(msg, "out");
    }
}

void Txc1::handleMessage(cMessage *msg)
{
    delete msg;
    if (strcmp("tic", getName()) == 0) {
        cMessage *newmsg = new cMessage("Hello, I am Tic. How are you, Toc!");
        send(newmsg, "out");
    }
    else {
        cMessage *msg = new cMessage("Hello, I am Toc. How are you, Tic!");
        send(newmsg, "out");
    }
}
```

kết quả:



8. Giả sử có thể phân tách tên đầy đủ của SV thành hai từ ABC và XYZ (ví dụ nếu tên SV là “Nguyen Van An” then ABC=”Van” and XYZ=”An”), chỉnh sửa mã nguồn hai module *tic/toc* sao cho chúng có tên là ABC và XYZ. Thay đổi mã nguồn sao cho các module gửi lời chào nhau với đúng tên của chúng trong câu (như ở 7).

sửa tệp tictoc1.ned:

```
simple Txc1
{
    gates:
        input in;
        output out;
}

network Tictoc1
{
    submodules:
        huy: Txc1;
        hoang: Txc1;
    connections:
        huy.out --> {  delay = 100ms; } --> hoang.in;
        huy.in <-- {  delay = 100ms; } <-- hoang.out;
}
```

sửa hai hàm trong tệp txc1.cc:

```
void Txc1::initialize()
{
    if (strcmp("huy", getName()) == 0) {
        cMessage *msg = new cMessage("Hello, I am Huy. How are you, Hoang!");
        send(msg, "out");
    }
}

void Txc1::handleMessage(cMessage *msg)
{
    delete msg;
```

```
    if (strcmp("huy", getName()) == 0) {
        cMessage *newmsg = new cMessage("Hello, I am Huy. How are you, Hoang!");
        send(newmsg, "out");
    }
    else {
        cMessage *newmsg = new cMessage("Hello, I am Hoang. How are you, Huy!");
        send(newmsg, "out");
    }
}
```

kết quả:

9. Giải thích ý nghĩa của câu lệnh sau: scheduleAt(simTime()+timeout,timeoutEvent);

Hàm scheduleAt(simtime_t t, cMessage *msg) là hàm lên lịch tự nhắn tin. Nó sẽ gửi trở lại mô-đun thông điệp msg thông qua hàm receive() hoặc handleMessage() tại thời điểm mô phỏng t.

Ý nghĩa của câu lệnh trên:

Trong thực tế, các thông điệp được gửi đi có thể bị lỗi hoặc bị mất do đường truyền hoặc thiết bị. Vì thế phía gửi phải có một cách thức để nhận biết việc gói tin có được gửi đi thành công hay không. Phía gửi sẽ ước tính trước một khoảng thời gian tối đa cho việc gửi và nhận thông điệp "nhận thành công" từ phía nhận. Nếu như quá khoảng thời gian này, coi như gói tin đã bị lỗi và cần phải gửi lại. Phía gửi sẽ tự gửi cho chính mình một thông điệp "timeoutEvent" để nhận biết sự kiện này.

10. Đặt K là số lượng các ký tự trong tên đầy đủ của SV và L là số lượng ký tự trong họ của SV. Chỉnh sửa mã nguồn sao cho số lượng các tin nhắn gửi từ tic đến toc là K và thời gian delay giữa hai lần gửi thông điệp là L (seconds)

"Nguyễn Huy Hoàng"

K = 16 (tính cả khoảng trắng)

L = 6

file txc1.cc:

```
#include <stdio.h>
#include <string.h>
#include <omnetpp.h>

using namespace omnetpp;
```

```
class Txc1 : public cSimpleModule
{
  private:
    cMessage *event;  // pointer to the event object which we'll use for timing
    cMessage *tictocMsg;  // variable to remember the message until we send it back
    int counter;

  public:
    Txc1();
    virtual ~Txc1();

  protected:
    virtual void initialize() override;
    virtual void handleMessage(cMessage *msg) override;
};

Define_Module(Txc1);

Txc1::Txc1()
{
    event = tictocMsg = nullptr;
}

Txc1::~Txc1()
{
    cancelAndDelete(event);
    delete tictocMsg;
}

void Txc1::initialize()
{
    counter = par("limit");
    event = new cMessage("event");
    if (strcmp("tic", getName()) == 0) {
        tictocMsg = new cMessage("tictocMsg");
        send(tictocMsg, "out");
    }
}

void Txc1::handleMessage(cMessage *msg)
{
    if (msg == event) {
        send(tictocMsg, "out");
        tictocMsg = nullptr;
    }
    else {
        counter--;
        if(counter == 0){
            return;
        }
        tictocMsg = msg;
        scheduleAt(simTime()+6.0, event); // L = 6
    }
}
```

file tictoc1.ned:

```
simple Txc1
{
    parameters:
        int limit = default(16); // K = 16
        @display("i=block/routing");
    gates:
        input in;
        output out;
```

```
}

network Tictoc1
{
    submodules:
        tic: Txc1 {
            parameters:
                @display("i=,cyan");
        }
        toc: Txc1 {
            parameters:
                @display("i=,gold");
        }
    connections:
        tic.out --> {  delay = 100ms; } --> toc.in;
        tic.in <-- {  delay = 100ms; } <-- toc.out;
}
```

kết quả: