گزارش فاز اول پروژه ۳ شبکه

محمدعلی قهاری ۸۱۰۱۰۰۲۰۱
رضا عبدلی ۸۱۰۱۰۰۲۵۱

توضیح کد:

```
void
readJson()
{
    QString val;
    QFile   file;
    file.setFileName("assets/config.json");
    file.open(QIODevice::ReadOnly | QIODevice::Text);
    val = file.readAll();
    file.close();
    QJsonDocument d     = QJsonDocument::fromJson(val.toUtf8());
    QJsonObject   sett2 = d.object();
    QJsonValue    value = sett2.value(QString("appName"));
    qWarning() << value;
    QJsonObject item = value.toObject();
    // qWarning() << tr("QJsonObject of description: ") << item;

    /* in case of string value get value and convert into string*/
    // qWarning() << tr("QJsonObject[appName] of description: ") << item["description"];
    QJsonValue subobj = item["description"];
    qWarning() << subobj.toString();

    /* in case of array get array and convert into string*/
    // qWarning() << tr("QJsonObject[appName] of value: ") << item["imp"];
    QJsonArray test = item["imp"].toArray();
    qWarning() << test[1].toString();
}
```

(برای این فاز نیست)در این قطعه کد فایل کانفیگ در assets/config.jsonرا میخوانیم و محتوای آن را به یک ابجکت JSON تبدیل می‌کنیم. سپس مقدار کلید `appName` را استخراج میکنیم. در ادامه از داخل appName، مقدار description و از آرایه impدرون appName (ایندکس ۱) را به‌عنوان یک رشته پرینت میکنیم.

```h
#ifndef PACKET_H
#define PACKET_H

#include "Globals.h"
#include "src/IP/IP.h"

#include <QObject>

class Packet : public QObject
{
    Q_OBJECT

public:
    explicit Packet(QObject *parent = nullptr);

private:
    UT::PacketType          type;
    uint                    waiting_cycles;
    uint                    total_cycles;
    QByteArray              payload;
    std::vector<AbstractIP> path;
    UT::TcpHeader           tcpHeader;
    void                   *ipHeader;
    UT::DataLinkHeader      dataLinkHeader;

Q_SIGNALS:
};

typedef QSharedPointer<Packet> PacketPtr_t;

#endif    // PACKET_H
```

در این قطعه کد هدر های لازم رو گذاشتیم که در فاز های بعدی باید ازش استفاده کنیم

```cpp
#include "macaddressgenerator.h"

MacAddressGenerator::MacAddressGenerator(QObject *parent) :
    QObject {parent}
{
    std::srand(810'100'201);
}

char *
MacAddressGenerator::generate()
{
    char *res = new char[12];
    for(int i = 0; i < 12; ++i)
        res[i] = std::rand() % 16;
    return res;
}
```

این قطعه کد میاد دیتای لازم برای تولید مک ادرس رو تولید میکنه

macaddress.cpp | MacAddress::MacAddress(char *, QObject *)

```cpp
#include "macaddress.h"

#include <iostream>
#include <sstream>

MacAddress::MacAddress(char address[12], QObject *parent) :
    QObject {parent}
{
    for(int i = 0; i < 12; ++i)
    {
        char c = address[i];
        for(int j = 3; j >= 0; --j)
        {
            if(c & 0b0001) this->address.set(47 - 4 * i - j);
            c >>= 1;
        }
    }
}

std::string
MacAddress::to_string()
{
    auto        divider = std::bitset<48>(15);
    std::string res     = "";
    for(auto i = 0; i < 12; i++)
    {
        std::stringstream temp;
        temp << std::hex << std::uppercase << ((address >> (4 * i)) & divider).to_ulong();
        res = temp.str() + res;
        if(i % 2 == 1 && i != 11) res = ':' + res;
    }
    return res;
}
```

در اینجا هم میاد آدرس MAC را به صورت بیتی ذخیره میکنه و سپس در متد `to_string` آن را به رشته‌ای با فرمت استاندارد MAC مثل (`XX:XX:XX:XX:XX:XX`) تبدیل و ریترن میکنه.

IP.cpp:

این کد کلاس‌هایی برای نمایش و مدیریت آدرس‌های IP در نسخه‌های IPv4 و IPv6 تعریف میکند.

در سازنده‌های کلاس IPv4 ادرس رو مقداردهی میکنه. آدرس‌ها به صورت بیت‌به‌بیت در یک std::bitset ذخیره می‌شوند.

متد to_string آدرس IPv4 را به صورت رشته‌ای با فرمت استاندارد (مثل ۱۹۲/۱۶۸/۱/۱) تبدیل می‌کند.

برای IPv6 نیز سازنده‌ها آدرس را از رشته (در قالب هگزادسیمال) یا مقادیر پیش‌فرض دریافت می‌کنند.

مشابه IPv4، متد to_string برای IPv6 آدرس را به رشته هگزادسیمال با قالب استاندارد تبدیل می‌کند.

EventsCoordinator.cpp | EventsCoordinator::instancePtr

```cpp
#include "EventsCoordinator.h"

#include <iostream>
#include <mutex>

#include <QTimer>

EventsCoordinator *EventsCoordinator::instancePtr = nullptr;
std::mutex         EventsCoordinator::mtx;
unsigned long long EventsCoordinator::num;
DataGenerator      EventsCoordinator::dataGenerator(Distribution::Poisson);

EventsCoordinator::EventsCoordinator(QThread *parent) :
    QThread {parent}
{}

void
EventsCoordinator::init(int millis)
{
    QTimer *timer = new QTimer();
    instance();
    connect(timer, &QTimer::timeout, instance(), &EventsCoordinator::clock);
    timer->start(Millis(millis));
}

void
EventsCoordinator::clock()
{
    auto data = EventsCoordinator::dataGenerator.generate(num++);
    std::cout << "call: " << data << std::endl;
    Q_EMIT global_tick(data);
}

EventsCoordinator *
EventsCoordinator::instance()
{
    if(instancePtr == nullptr)
    {
        std::lock_guard<std::mutex> lock(mtx);
        if(instancePtr == nullptr) instancePtr = new EventsCoordinator();
    }
    return instancePtr;
}
```

در اینجا هم میاد یه جنریتور میسازه موقع اینیت بعد یه تایمر میزاره که از ارگومان گرفته بعد هی داخل کلاک سیگنالش رو با توجه به دیتایی که از دیتا جنریتور گرفته emit میکنه (به صورت دوره‌ای با استفاده ازQTimer)

```cpp
#include "datagenerator.h"

#include <random>

DataGenerator::DataGenerator(Distribution distr, QObject *parent) :
    QObject {parent}
{
    distribution = distr;
    gen          = std::mt19937(std::random_device()());
    poisson      = std::poisson_distribution<>(4);
}

int
DataGenerator::generate(int i)
{
    if(distribution == Distribution::Poisson)
    {
        return poisson(gen);
    }
    else
    {
        return 1;
    }
}
```

در اینجاهم با توزیع پواسون داده تصادفی ایجاد میکنه