3/10/2022

Hw2 report

Cryptocurrencies

Mohammadreza babaei mosleh : 9823011

دانشگاه صنعتی امیرکبیر

در این گزارش به پیاده سازی ابتدایی عملکرد مالی رمز ارز ها میپردازیم و با پیاده سازی توابع مختلف مجموعه ای ساده شده از استخراج و انتقال را به نمایش میذاریم این برنامه در غالب سه فایل: Server – crypto – Client نوشته شده است که در ان هر فایل درغالب دو زیر مجموعه cpp. و h. معرفی میشود.
در ادامه این گزارش ابتدا به بررسی اجمالی دو فایل Server و client میپردازیم و در ادامه به بررسی توابع و نحوه عملکرد ان ها میپردازیم و در صورت نیاز به بیان نکات مهم و اشتباهات پرتکرار و مشکلات پیش امده برایمان می پردازیم

:Server

همانطور که گفته شد این فایل دارای دو زیر مجموعه cpp. و h. میباشد که فایل h. محتوی declarationها و فایل cpp. محتوی definition ها میباشد این فایل در ابتدای برنامه فرا خوانده میشود و یک سرور را ایجاد میکنند که در بخش بعدی client ها بتوانند از ان استفاده کنند.

```
#ifndef SERVER_H
#define SERVER_H

#include <memory>
#include <string>
#include <map>
#include "client.h"
#include <crypto.h>
#include <random>
#include <vector>

//defining client for avoinding redefinition
class Client;
// telling compiler that this variable is defined somewhere in server.cpp
extern std::vector<std::string> pending_trxs;

class Server
{

public:
    Server();
    std::shared_ptr<Client> add_client(std::string id);
    std::shared_ptr<Client> get_client(std::string id) const;
```

```
    double get_wallet(std::string id) const;
    static bool parse_trx(std::string trx, std::string &sender, std::string
&receiver, double &value);
    bool add_pending_trx(std::string trx, std::string signature) const;
    size_t mine();
    //accessing to private varialbe of class from out of it
    friend void  show_wallets(const  Server& server);

private:
    std::map<std::shared_ptr<Client>, double> clients;

};




#endif //SERVER_H
```

همانطور که میبینید دارای method های متعدد میباشد که در ادامه توضیح داده میشود .

نکته :با توجه به اینکه کلاس های client و server در داخل هم به صورت تو در تو استفاده شده اند برای اینکه داخل loop اگیر نکنیم و بتوانیم از ان ها به این شکل استفاده کنیم تعریفی یک خطی از هرکدام را مانند زیر به اول کد ها افزوده ایم.

```
class Client;
```

```
class Server;
```

:Client

این فایل نیز دارای دو زیر مجموعه دارای تعاریف بالا میباشد و با استفاده از ان میتوان client های را ایجاد کرد و با اتصال ان ها به سرور ان ها را به یکدیگر متصل کرد و در فرایند استخراج توسط server و همچنین انتقال داده ها شرکت داد.

```
#ifndef CLIENT_H
#define CLIENT_H

#include <memory>
#include <string>
#include "server.h"
#include <string>
#include <crypto.h>
#include <random>
#include <vector>

//defining class for avoiding redefinition error
class Server;

class Client
{

public:
    Client(std::string _id, const Server& _server);
    std::string get_id();
    std::string get_publickey() const;
    double get_wallet();
    std::string sign(std::string txt) const;
    bool transfer_money(std::string receiver, double value);
    size_t generate_nonce();

private:
    Server const* const server;
    const std::string id;
    std::string public_key;
    std::string private_key;

};

#endif //CLIENT_H
```

همانطور که مشاهده میکنید دارای method های متعدد هست که در ادامه بحث میشوند و همچنین چند متغیر به صورت private دارد که ب نوبه خود تشریح خواهند شد و به ان ها پرداخته میشود.

:Method’s

```
Server::Server()
{
}
```

این method در واقع constructor method برای Server میباشد و object های ما با استفاده از ان ساخته میشوند همانطور که میبینید ورودی نمیگیرد و تنها object مورد نظر را برای استفاده در ادامه فراخوانی میکند.

```
Client::Client(std::string _id, const Server& _server)
    : server{ &_server }
    , id{ _id }

{

    crypto::generate_key(this->public_key,this->private_key);

}
```

این method در واقع constructor method برای Client می باشد و object ها با استفاده از ان ساخته میشوند.

نکته قابل توجه در مورد این method این است که با توجه به اینکه متغیر server که در کلاس تعریف شده است به صورت const* const میباشد این به این معناست که متغیر از نوع اشاره گری const است که به خانه ای از حافظه اشاره میکند که محتویات ان نیز const هستند پس برای اینکه به مشکل بر نخوریم این متغیر ها تنها یکبار ان هم در موقع به وجود امدن object باید مقدار دهی شوند که با syntax بالا همخوانی دارد.

در مورد متغیر id هم به همین شکل است چون const است باید در موقع ساخته شدن مقدار دهی شود. درادامه با استفاده از توابع از پیش ساخته شده در فایلcrypto به مقدار دهی متغیر های public_key و private_key میپردازد

```
// adding a client to the server
std::shared_ptr<Client> Server::add_client(std::string id)
{
    // iterator on map
    std::map<std::shared_ptr<Client>, double>::iterator itr;

    for(itr = clients.begin(); itr != clients.end(); ++itr)
    {
        // adding 4 random digit to id
        if(itr->first->get_id() == id)
        {
            std::random_device rd;
            std::mt19937 mt(rd());
            std::uniform_int_distribution<int> dist(1000, 9999);
            id += std::to_string(dist(mt));
        }

    }
    //making new client
    Client client{id, *this};
    std::shared_ptr<Client> tmp{ std::make_shared<Client>(client) };

    //adding client to clients and set init value to 5
    clients.insert(std::pair<std::shared_ptr<Client>, double>(tmp, 5));

    return tmp;

}
```

این method متعلق به Server بوده و از ورودی یک id دریافت کرده و داخل همین سرور یک client با id دریافت شده را ایجاد میکند .نکته قابل توجه ای است که اگر id مورد نظر از قبل وجود داشت چهار رقم به صورت کاملا تصادفی به دنباله ان اضافه میکند و client مربوطه را ایجاد میکند در اخر یک اشاره گر از نوع shared_ptr از ان تولید کرده و با استفاده از ان client را در داخل لیست موجود از clientهای سرور با نام clients اضافه میکند و مقدار value را که در واقع مقدار دارایی ان client است را به لیست اضافه میکند .

در اخر به عنوان خروجی اشاره گر تولید شده از client را برمیگرداند

```
std::string Client::get_id()
{

    return id;

}
```

این method یک getter method برای متغیر id میباشد که با توجه به private بودن ان به ان نیاز است.

```
std::shared_ptr<Client> Server::get_client(std::string id) const
{

    for(auto itr : clients)
    {

        if(itr.first->get_id() == id)
        {

            return itr.first;

        }

    }

    return nullptr;
}
```

این method متغیر id را به عنوان ورودی دریافت میکند و با استفاده از ان در مجموعه clients جست و جو میکند و در صورت وجود ان مقدار shared_ptr ان را به عنوان خروجی برمیگرداند نکته جالب در این تابع نحوه کارکرد متغیر itr است که در واقع یک اشاره گر به object خود است که دارای دو متغیر first و second میباشد.

```
double Server::get_wallet(std::string id) const
{
    //accesssing to id
    std::shared_ptr<Client> tmp{ get_client(id) };

    //mapping pointer to value
    return clients.find(tmp)->second;
}
```

این method متغیر id را به عنوان ورودی دریافت کرده و با استفاده از get_client ابتدا به shared_ptr مربوط به object مورد نظر دسترسی پیدا میکند سپس پس از پیدا کردن ان در clients با صدا کردن second مربوط به ان به value مربوط به ان دسترسی پیدا میکند.

نکته دیگر وجود const در انتهای تعریف این method است که به دلیل این است که متغیر های خارجی که const هستند توانایی استفاده از ان را داشته باشند.

```
bool Server::parse_trx(std::string trx, std::string &sender, std::string
&receiver, double &value)
{
    int first_{} , second_{};
    // variable for extracted value
    std::string str_value{};

    //check if there is two '-' in trx
    if(trx.find('-') != std::string::npos)
    {

        first_ = trx.find('-');
        if(trx.find('-', first_ + 1) != std::string::npos)
        {
            //checking special cases
            second_ = trx.find('-', first_ + 1);


            if(first_ != 0)
            {

               sender = trx.substr(0, first_);
```

```
            }
            else
            {

                throw std::runtime_error("invalid trx format");

            }

            if(second_ - first_ != 1)
            {

                receiver = trx.substr(first_ + 1, second_ - first_ - 1);

            }
            else
            {

                throw std::runtime_error("invalid trx format");

            }

            if(second_ != trx.length() - 1)
            {

               str_value = trx.substr(second_ + 1, trx.length() - (second_) - 1
);

            }
            else
            {

                throw std::runtime_error("invalid trx format");

            }

            for(auto chr : str_value)
            {

                if (std::isdigit(chr) == 0 && chr != '.')
                {

                    throw std::runtime_error("invalid trx format");

                }
```

```
            }

            //storing value
            value = std::stod(str_value);
            return true;
        }
        else
        {

            throw std::runtime_error("invalid trx form");

        }

    }

    else
    {

        throw std::runtime_error("invalid trx form");

    }

}
```

این method که متعلق به server است با دریافت چهار متغیر trx, sender, reveiver, value را دریافت میکند نقش حیاتی در کار دارد و دو کار مهم را انجام میدهد اول انکه چک میکند که ایا trx ورودی به فرمت درست یعنیsender-reveiver-value هست یا نه در صورتی که جواب منفی بود runtime error برمیگرداند و در صورتی که format درست بود با parse کردن ان مقادیر sender,reveiver و value را پس از چک کردن validity مقدار دهی میکند با توجه به این که ورودی ها به صورت"input&" وارد شده اند در نتیجه این methodمیتواند مقادیر اصلی متغیر های ورودی را تغییر دهد.

```
std::string Client::get_publickey() const
{

    return public_key;

}
```

این methodکه متعلق به client است در واقع یک getter method است که با توجه به private بودن public_key برای دسترسی به ان مورد نیاز است

وجود const در انتها به دلیل توانایی فراخوانی ان با استفاده از متغیر هایconst است.

```
double Client::get_wallet()
{

    return server->get_wallet(id);

}
```

این method با وجه به private بودن مجموعه clients رد server به هر client اجازه میدهد با استفاده از methodی که با همین نام برای server وجود دارد به مقدار value خود دسترسی داشته باشند.

```
std::string Client::sign(std::string txt) const
{

    std::string signature = crypto::signMessage(private_key, txt);
    return signature;

}
```

این method که مربوط به client است با دریافت یک trx و با استفاده از توابع موجود در crypto یک signature تولید کرده و به اصطلاح تراکنش را امضا میکند و امضا را برمیگرداند.

وجودconst در انتهای تعریف به دلیل توانایی فراخوانی بااستفاده از متغیرهای const است

```
std::vector<std::string> pending_trxs;
//adding trxs to pending_trxs
bool Server::add_pending_trx(std::string trx, std::string signature) const
{

    std::string sender{} , receiver{};
    double value{};
    //parsing
    if(parse_trx(trx, sender, receiver, value))
    {
        //checking validity of trx and adding
        bool authentic = crypto::verifySignature(get_client(sender)-
>get_publickey(), trx, signature);
        double sender_money{ clients.find(get_client(sender))->second };
        bool value_check = (sender_money >= value);

        if(value_check && authentic)
        {

            pending_trxs.push_back(trx);
            return true;

        }

    }
    return false ;

}
```

این method با دریافت یک trx وsignature ابتدا چک میکند که ایا trx دریافت شده valid هست یا نه و در این صورت ان را parse کرده و اطلاعات را از ان استخراج میکند نکته جالب در مورد شکل بالا تعریف یک vector به صورتی است که خارج از کلاس server است مشکلی که در این مورد پیش میاید error است که به دلیل چندین بار فراخوانی server به وجود می اید و ان double definition یا redefinition است برای اینکه به این مشکل بر نخوریم باید همین تعریف را با پسوند extern به کلاس client نیز اضافه کنیم(به شکل مربوط به declaration ها دقت کنید)این کار باعث میشود که و که

Compiler به این خط رسید به ان اطلاع دهد که این متغیر جایی در بدنه تابع اصلی تعریف شده است و به ان دستور میدهد که دنبال ان بگردد و این کار از error مورد نظر جلوگیری میکند.

پس از تایید validity ورودی trx و استخراج اطلاعات ان با استفاده از توابعی که قبل تر تعریف شده چک میکند که ایا این تراکنش ممکن است یا نه یعنی ایا receiver وجود دارد و در صورت وجود ان ایا مبلغ کافی برای انجام تراکنش را دارد یا نه.

Constدر انتهایmethod برای توانایی استفاده از ان با متغیرهای const میباشد.

```
bool Client::transfer_money(std::string receiver, double value)
{

    std::string value_trx{ std::to_string(value) };
    if(server->get_client(receiver))//checking if receiver exists
    {

        std::string trx{ id + "-" + receiver+ "-" + value_trx };
        std::string signature{ sign(trx) };
        if(server->add_pending_trx(trx, signature))
        {

            return true;

        }
        else
        {

            return false;

        }

    }
    else
    {

        return false;

    }
```

این method که متعلق به کلاس Client است یک receiver و یک value دریافت میکند چک میکند در صورت وجود receiver تراکنش را به فرم استاندارد تعریف میکند و سپس با استفاده از متد sign ان را امضا میکند و سپس تراکنش را با استفاده از متد add_pending_trx که متعلق به سروری است که client روی ان ساخته شده به مجموعه تراکنش های در دست انجام یعنی pending_trx اضافه میکند .

```
size_t  Client::generate_nonce()
{
    //generating random number for nonce
    std::random_device rd;
    std::mt19937 mt(rd());
    std::uniform_int_distribution<size_t> dist;
    size_t nonce{ (dist(mt)) };
    return nonce;

}
```

این method که متعلق به client است با استفاده از الگوریتم تولید عدد تصادفی عدد را تولید کرده به عنوان nonce برای انجام عملیات mine به server برمیگرداند

```
size_t Server::mine()
{

    size_t flag{}; //flag for checking of seccessfull mine
    size_t winner_nonce{};
    size_t finding_nonce{ 1 }; //flag for checking valid nonce
    std::string winner_id{};
    std::string mempool{};
    //final string of apended trxs
    for(auto tmp : pending_trxs)
    {
```

```
        mempool += tmp;

    }

    while(finding_nonce)
    {
        //taking nonce from every eche client
        for(auto& client : clients)
        {

            std::string mempool_backup{ mempool }; //backup just in case of
neeeded
            size_t nonce{ client.first->generate_nonce() };
            mempool_backup += std::to_string(nonce);
            //hashing
            std::string hash{ crypto::sha256(mempool_backup) };
            //check for successfull
            if((hash.substr(0 , 10)).find("000") != std::string::npos)
            {

                client.second += 6.25; //reward
                winner_id = client.first->get_id();
                winner_nonce = nonce;
                flag = 1;
                finding_nonce = 0;
                break;

            }

        }

    }

    if(flag)
    {

        for(auto trx : pending_trxs)
        {

            std::string sender{};
            std::string receiver{};
            double value{};
            parse_trx(trx, sender, receiver, value);
            clients.find(get_client(sender))->second -= value;
            clients.find(get_client(receiver))->second += value;
```

```
        }

        pending_trxs.clear();//clearing pending_trxs
        std::cout << winner_id << std::endl;
        return winner_nonce;

    }
    else
    {

        return static_cast<size_t>(NULL);

    }

}
```

این method که مربوط به server است در واقع ماه عسل تمام توابع بالا است به این صورت که ابتدا با دسترسی به مجموعه pending_trxs همه اعضای ان را به صورت متوالی در متغیر mempool به هم concatenate میکند و یک string از همه تراکنش ها میسازد سپس از با استفاده از حلقه for روی clients از هر client میخواهد که با استفاده از nonce generating , method یک nonce را به صورت random به server ارسال کند. سپس پس از concatenate کردن mempool و nonce ان را به تابع hash از فایلcrypto میدهد و sha256 hash ان را بررسی میکند در صورتی که در داخل hash و در داخل 10 حرف اول ان ‘000’ وجود داشت ان client برنده را save میکند اینکار را انقدر انجام میدهد که یکی از client ها به nonce درست برسد و مقدار reward را به client برنده اضافه میکند

پس از ان به سراغ مجموعه pending_trx میرود و با استفاده از for یک به یک به تمام اعضا دسترسی پیدا میکند و با استفاده از parse کردن ان ها با method مربوطه data های مورد نیاز را از ان استخراج میکند و تراکنش مربوط به هر دیتا را انجام میدهد

پس از انجام این کار pending_trx را پاک و برنده را اعلام میکند nonce برنده را برمیگرداند

پایان توضیح method ها .

:Questions

1 – پس از قرار دادن تابع خواسته شده در main.cpp طبق خواسته سوال و درست کردن test case مورد نظر دیدم که نتیجه مورد انتظار است.

```
********************
mamad-mahdi-3.000000
mahdi-mamad-2.000000
********************
```

2 – پس از قرار دادن تابع زیر در server:

```
void  show_wallets(const  Server& server)
{

    std::cout << std::string(20, '*') << std::endl;
    for(const auto& client: server.clients)
    std::cout << client.first->get_id() <<  " : "  << client.second << std::endl;
    std::cout << std::string(20, '*') << std::endl;

}
```

با توجه به اینکه تابع برای اینکه درست کار کند نیاز داد به مجموعه clients دسترسی داشته و باشد و clients به صورت private در Server تعریف شده است با استفاده از دستور friend مطابق دستور زیر این کار را انجام میدهیم.

```
friend void  show_wallets(const  Server& server);
```

THE END

Github link : https://github.com/ghostoftime111/hw2.git