Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №6-8 по курсу**

**«Операционные системы»**

**Тема работы**

**Управлении серверами сообщений.**

**Применение отложенных вычислений.**

**Интеграция программных систем друг с другом.**

Студент: Чистяков Д. М.

Группа: М8О-206Б-20

Вариант: 46

Преподаватель: Соколов Андрей Алексеевич

Оценка:

Дата: 02.04.22

Подпись: \_\_\_\_\_\_\_\_\_\_

Москва, 2022

**Содержание**

1. Постановка задачи
2. Общие сведения о программе
3. Исходный код
4. Демонстрация работы программы
5. Выводы

**Постановка задачи**

Реализовать распределенную систему по асинхронной обработке запросов. В данной распределенной системе существует 2 вида узлов: «управляющий» и «вычислительный». Узлы объединены в топологию «дерево общего вида». Связь между узлами необходимо осуществить при помощи технологии очередей сообщений. Также в данной системе необходимо предусмотреть проверку доступности узлов в соответствии с вариантом. При убийстве («kill -9») любого вычислительного узла система должна пытаться максимально сохранять свою работоспособность, а именно все дочерние узлы убитого узла могут стать недоступными, но родительские узлы должны сохранить

свою работоспособность.

Управляющий узел отвечает за ввод команд от пользователя и отправку этих команд на вычислительные узлы.

**Общие сведения о программе**

Файлы TopologyNode.hpp и TopologyTree.hpp содержат реализации узла топологии и самой топологии, файлы control.cpp и calc.cpp содержат реализацию управляющего и вычислительного узла, сборка осуществляется с помощью утилиты make.

**Исходный код**

Makefile:

all: control calc

control:

g++ control.cpp -lzmq -o control.out -Wall

calc:

g++ calc.cpp -lzmq -o calc.out -Wall

clean:

rm -rf control.out calc.out

TopologyNode.hpp:

#ifndef \_\_TOPOLOGY\_NODE\_HPP\_\_

#define \_\_TOPOLOGY\_NODE\_HPP\_\_

#include <zmq.hpp>

class TopologyNode

{

public:

    int id;

    TopologyNode\* parent;

    TopologyNode\* oldestChild;

    TopologyNode\* rightBro;

    bool removed;

    zmq::socket\_t socket;

    TopologyNode(TopologyNode\* parent, int id)

    {

        this->parent = parent;

        this->oldestChild = nullptr;

        this->rightBro = nullptr;

        this->id = id;

        removed = false;

    }

};

#endif

TopologyTree.hpp:

#ifndef \_\_TOPOLOGY\_TREE\_HPP\_\_

#define \_\_TOPOLOGY\_TREE\_HPP\_\_

#include "TopologyNode.hpp"

class TopologyTree

{

public:

    TopologyNode\* root;

    TopologyTree()

    {

        root = new TopologyNode(nullptr, -1);

    }

    TopologyNode\* findNode(TopologyNode\* curNode, int id)

    {

        if(curNode != nullptr)

        {

            TopologyNode\* ans = nullptr;

            if(!curNode->removed)

            {

                ans = findNode(curNode->oldestChild, id);

            }

            if(ans != nullptr && !ans->removed)

            {

                return ans;

            }

            if(curNode->id == id && !curNode->removed)

            {

                ans = curNode;

                return ans;

            }

            return findNode(curNode->rightBro, id);

        }

        else

        {

            return nullptr;

        }

    }

    TopologyNode\* addNode(int nodeId, int parentId)

    {

        TopologyNode\* parent = findNode(root, parentId);

        TopologyNode\* node = new TopologyNode(parent, nodeId);

        if(parent->oldestChild == nullptr)

        {

            parent->oldestChild = node;

        }

        else

        {

            TopologyNode\* leftBro = parent->oldestChild;

            while(leftBro->rightBro != nullptr)

            {

                leftBro = leftBro->rightBro;

            }

            leftBro->rightBro = node;

        }

        return node;

    }

    void \_recursClear(TopologyNode\* curNode)

    {

        if(curNode != nullptr)

        {

            \_recursClear(curNode->oldestChild);

            \_recursClear(curNode->rightBro);

            curNode->socket.close();

            delete curNode;

        }

    }

    void clear()

    {

        \_recursClear(root->oldestChild);

        root->oldestChild = nullptr;

    }

    void removeNode(int nodeId)

    {

        TopologyNode\* deletedNode = findNode(root, nodeId);

        deletedNode->socket.close();

        deletedNode->removed = true;

    }

    void recursivePrint(TopologyNode\* curItem)

    {

        if (curItem != nullptr)

        {

            std::cout << curItem->id;

            if(curItem->removed)

            {

                std::cout << "d";

            }

            if (curItem->oldestChild != nullptr)

            {

                std::cout << " : [";

            }

            recursivePrint(curItem->oldestChild);

            if (curItem->rightBro != nullptr)

            {

                std::cout << ", ";

            }

            recursivePrint(curItem->rightBro);

            if (curItem->rightBro == nullptr)

                std::cout << "]";

        }

    }

    void print()

    {

        std::cout << root->id;

        std::cout << " : [";

        recursivePrint(root->oldestChild);

    }

};

#endif

calc.cpp:

#include <iostream>

#include <zmq.hpp>

#include <map>

zmq::context\_t context;

int main(int argc, const char\* argv[])

{

    char DELIM = '|';

    zmq::socket\_t socket(context, zmq::socket\_type::rep);

    socket.bind(argv[0]);

    std::map<std::string, int> dict;

    bool exist = true;

    while(exist)

    {

        zmq::message\_t zIn;

        if(socket.recv(zIn).value() == -1ul)

        {

            return 1;

        }

        std::string msgIn = zIn.to\_string();

        std::string command = msgIn.substr(0, msgIn.find\_first\_of(DELIM));

        if(command == "ping")

        {

            std::string msgOut("Ok: 1");

            zmq::message\_t zOut(msgOut);

            socket.send(zOut, zmq::send\_flags::none);

        }

        else if(command == "exec")

        {

            int delimAmount = std::count(msgIn.begin(), msgIn.end(), DELIM);

            std::string msgOut("Ok:");

            if(delimAmount == 2)

            {

                int id = std::stoi(msgIn.substr(msgIn.find\_first\_of(DELIM) + 1,

                                   msgIn.find\_last\_of(DELIM) - msgIn.find\_first\_of(DELIM) - 1));

                msgOut += std::to\_string(id);

                std::string key = msgIn.substr(msgIn.find\_last\_of(DELIM) + 1);

                if(dict.find(key) == dict.end())

                {

                    msgOut += ": '" + key + "' not found";

                }

                else

                {

                    msgOut += ": " + std::to\_string(dict[key]);

                }

            }

            else

            {

                int secDelimPos = msgIn.find(DELIM, msgIn.find\_first\_of(DELIM) + 1);

                int id = std::stoi(msgIn.substr(msgIn.find\_first\_of(DELIM) + 1,

                                   secDelimPos - msgIn.find\_first\_of(DELIM) - 1));

                msgOut += std::to\_string(id);

                std::string key = msgIn.substr(secDelimPos + 1, msgIn.find\_last\_of(DELIM) - secDelimPos - 1);

                int value = std::stoi(msgIn.substr(msgIn.find\_last\_of(DELIM) + 1));

                dict[key] = value;

            }

            zmq::message\_t zOut(msgOut);

            socket.send(zOut, zmq::send\_flags::none);

        }

        else if(command == "kill")

        {

            exist = false;

        }

    }

    socket.close();

    return 0;

}

control.cpp:

#include <iostream>

#include <vector>

#include <zmq.hpp>

#include <unistd.h>

#include "TopologyTree.hpp"

#include <set>

zmq::context\_t context;

int main()

{

    TopologyTree\* topologyTree = new TopologyTree();

    std::set<int> existingNodes;

    std::string command;

    int id;

    std::cout << "> ";

    while(std::cin >> command){

        if(command == "ping")

        {

            std::cin >> id;

            TopologyNode\* node = topologyTree->findNode(topologyTree->root, id);

            if(node == nullptr && existingNodes.find(id) == existingNodes.end())

            {

                std::cout << "Error: Not found\n";

            }

            else if(node == nullptr && existingNodes.find(id) != existingNodes.end())

            {

                std::cout << "Ok: 0\n";

            }

            else

            {

                std::string msgOut = command + "|" + std::to\_string(id);

                zmq::message\_t zOut(msgOut);

                node->socket.send(zOut, zmq::send\_flags::none);

                zmq::message\_t zIn;

                zmq::recv\_result\_t rc = node->socket.recv(zIn);

                if(rc.value() == -1ul)

                {

                    return 1;

                }

                else if(!rc.has\_value())

                {

                    existingNodes.erase(id);

                    topologyTree->removeNode(id);

                }

                else

                {

                    std::cout << zIn.to\_string() << std::endl;

                }

            }

        }

        else if(command == "create")

        {

            std::cin >> id;

            int parentId;

            std::cin >> parentId;

            if(existingNodes.find(id) != existingNodes.end())

            {

                std::cout << "Error: Already exists\n";

            }

            else if(topologyTree->findNode(topologyTree->root, parentId) == nullptr

                    && existingNodes.find(parentId) == existingNodes.end())

            {

                std::cout << "Error: Parent not found\n";

            }

            else if(topologyTree->findNode(topologyTree->root, parentId) == nullptr

                    && existingNodes.find(parentId) != existingNodes.end())

            {

                std::cout << "Error: Parent is unavailable\n";

            }

            else

            {

                TopologyNode\* node = topologyTree->addNode(id, parentId);

                node->socket = zmq::socket\_t(context, zmq::socket\_type::req);

                node->socket.setsockopt(ZMQ\_RCVTIMEO, 2000);

                const std::string addr = "tcp://127.0.0.1:" + std::to\_string(5555 + id);

                node->socket.connect(addr);

                existingNodes.insert(id);

                topologyTree->print();

                std::cout << std::endl;

                int pid = fork();

                if(pid == 0)

                {

                    execl("calc.out", addr.c\_str(), NULL);

                }

                else if(pid > 0)

                {

                    std::cout << "Ok: " << pid << "\n";

                }

            }

        }

        else if(command == "exec")

        {

            std::cin >> id;

            TopologyNode\* node = topologyTree->findNode(topologyTree->root, id);

            if(node == nullptr && existingNodes.find(id) == existingNodes.end())

            {

                std::cout << "Error: Not found\n";

            }

            else if(node == nullptr && existingNodes.find(id) != existingNodes.end())

            {

                std::cout << "Error: Node is unavailable\n";

            }

            else

            {

                std::string key;

                std::cin >> key;

                int value;

                std::string msgOut;

                if(getchar() == ' ')

                {

                    std::cin >> value;

                    msgOut = command + "|" + std::to\_string(id) + "|" + key + "|" + std::to\_string(value);

                }

                else

                {

                    msgOut = command + "|" + std::to\_string(id) + "|" + key;

                }

                zmq::message\_t zOut(msgOut);

                node->socket.send(zOut, zmq::send\_flags::none);

                zmq::message\_t zIn;

                if(node->socket.recv(zIn).value() == -1ul)

                {

                    return 1;

                }

                std::cout << zIn.to\_string() << std::endl;

            }

        }

        else if(command == "kill")

        {

            std::cin >> id;

            TopologyNode\* node = topologyTree->findNode(topologyTree->root, id);

            std::string msgOut = command + "|" + std::to\_string(id);

            zmq::message\_t zOut(msgOut);

            node->socket.send(zOut, zmq::send\_flags::none);

            existingNodes.erase(id);

            topologyTree->removeNode(id);

        }

        else if(command == "exit")

        {

            break;

        }

        else

        {

            std::cout << "Error: Unknown command\n";

        }

        std::cout << "> ";

    }

    topologyTree->clear();

    delete topologyTree;

    return 0;

}

**Демонстрация работы программы**

den@den-VirtualBox:~/Документы/lab6$ make

g++ control.cpp -lzmq -o control.out -Wall

g++ calc.cpp -lzmq -o calc.out -Wall

den@den-VirtualBox:~/Документы/lab6$ ./control.out

> create 3 -1

-1 : [3]

Ok: 1888

> create 1 -1

-1 : [3, 1]

Ok: 1891

> create 2 -1

-1 : [3, 1, 2]

Ok: 1895

> create 2 3

Error: Already exists

> create 4 3

-1 : [3 : [4], 1, 2]

Ok: 1907

> create 5 4

-1 : [3 : [4 : [5]], 1, 2]

Ok: 1910

> exec 3 M 33

Ok:3

> exec 3 M

Ok:3: 33

> exec 3 F

Ok:3: 'F' not found

> kill 2

> ping 1

Ok: 1

> ping 4

Ok: 1

> kill 4

> ping 4

Error: Not found

> ping 5

Ok: 0

> ping 3

Ok: 1

> exec 3 M 44

Ok:3

> exec 3 M

Ok:3: 44

> exit

**Выводы**

В ходе выполнения лабораторной работы я научился использовать очередь сообщений и освоил технологию ZeroMQ.