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: Полонский Кирилл Андреевич
Группа: М8О-208Б-20
Вариант: 46
Преподаватель: Миронов Евгений Сергеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2021

Содержание

Репозиторий

https://github.com/kirillpolonskii/OS/tree/master/os_lab6

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 topology_node.hpp и topology_tree.hpp содержат реализации узла топологии и самой топологии, файлы control_node.cpp и calc_node.cpp содержат реализацию управляющего и вычислительного узла, сборка осуществляется с помощью утилиты cmake.

Общий метод и алгоритм решения

Исходный код

topology_node.hpp

```
#ifndef OS_LAB6_CLION_TOPOLOGY_NODE_HPP
#define OS_LAB6_CLION_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 //OS_LAB6_CLION_TOPOLOGY_NODE_HPP
```

topology_tree.hpp

```
#ifndef OS_LAB6_CLION_TOPOLOGY_TREE_HPP
#define OS_LAB6_CLION_TOPOLOGY_TREE_HPP

#include "topology_node.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, TopologyNode* deletedNode){// позже надо, наверное, переписать
        if(curNode != nullptr){
            recursClear(curNode->oldestChild, deletedNode);
            if(curNode != deletedNode){
                recursClear(curNode->rightBro, deletedNode);
                curNode->socket.close();
            }
            delete curNode;
        }
    }

    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 ";
            }
            else{
                std::cout << " ";
            }
            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(){
        recursivePrint(root);
    }
};
#endif //OS_LAB6_CLION_TOPOLOGY_TREE_HPP
```

control_node.cpp

```
#include <iostream>
#include <vector>
#include <zmq.hpp>
#include <unistd.h>
#include "topology_tree.hpp"
#include <set>
//this is a client side

zmq::context_t context;

int main() {
    TopologyTree* topologyTree = new TopologyTree();
    std::set<int> existingNodes;

    std::string command;
    int id;

    while(std::cin >> command >> id){
        if(command == "ping"){
            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 == -1){
                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"){
        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_node", addr.c_str(), NULL);
        }
        else if(pid > 0){
            std::cout << "Ok: " << pid << "\n";
        }
    }

}
else if(command == "exec"){
    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() == ' '){ // request for adding a value
            std::cin >> value;
            msgOut = command + "|" + std::to_string(id) + "|" + key + "|" + std::to_string(value);
        }
```

```
            else{ // request for loading a value
                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) == -1){
                return 1;
            }
            std::cout << zIn.to_string() << std::endl;

        }
    }
    else if(command == "kill"){
        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;
    }
  }
  topologyTree->recursClear(topologyTree->root, topologyTree->root);
  delete topologyTree;
  return 0;
}
```

calc_node.cpp

```
#include <iostream>
#include <zmq.hpp>
```

```
#include <map>
//this is a server side


zmq::context_t context;

int main(int argc, const char* argv[]) {
    char DELIM = '|';
    zmq::socket_t socket(context, zmq::socket_type::rep);
    //socket.setsockopt(ZMQ_SNDTIMEO, 4000);
    socket.bind(argv[0]);
    std::map<std::string, int> dict;

    bool exist = true;
    while(exist){
        zmq::message_t zIn;
        if(socket.recv(zIn) == -1){
            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){ // request for loading a value
                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{ // request for adding a value
            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;
}
```

Демонстрация работы программы

```
kirill@kirill-acpire:~/labsMAI/sem3/os_lab6_clion/cmake-build-debug$
./control_node
create 3 -1
-1 : [3 ]]
Ok: 14567
create 1 -1
-1 : [3 , 1 ]]
```

```
Ok: 14570
create 2 -1
-1 : [3 , 1 , 2 ]]
Ok: 14574
create 4 3
-1 : [3 : [4 ], 1 , 2 ]]
Ok: 14584
create 5 4
-1 : [3 : [4 : [5 ]], 1 , 2 ]]
Ok: 14587
exec 3 M 90
Ok:3
exec 3 P
Ok:3: 'P' not found
exec 3 M
Ok:3: 90
kill 2
ping 1
Ok: 1
ping 4
ping 4
Error: Not found
ping 5
Ok: 0
ping 3
Ok: 1
exec 3 M 67
Ok:3
exec 3 M
Ok:3: 67
```

Выводы

В ходе выполнения лабораторной работы я научился использовать очередь сообщений.