Московский авиационный институт
(Национальный исследовательский университет)
Факультет "Информационные технологии и прикладная математика"
Кафедра "Вычислительная математика и программирование"

Лабораторная работа №6 по курсу “Операционные системы”

Студент: Живалев Е.А.

Группа: М8О-206Б

Преподаватель: Соколов А.А.

Вариант: 26

Оценка: _____

Дата: _____

Подпись: _____

Москва, 2019

1 Задание

Реализовать распределенную систему по асинхронной обработке запросов. В данной распределенной системе должно существовать 2 вида узлов: «управляющий» и «вычислительный». Необходимо объединить данные узлы в соответствии с той топологией, которая определена вариантом. Связь между узлами необходимо осуществить при помощи технологии очередей сообщений. Также в данной системе необходимо предусмотреть проверку доступности узлов в соответствии с вариантом.

Тип топологии: Дерево общего вида Тип команды: Локальный таймер Тип проверки доступности узлов: Пинг узла с указанным id

2 Описание работы программы

Программа разбита на файлы socketRoutine.hpp, socketRoutine.cpp(отвечают за работу с сокетами - получение и отправка сообщений, создание сокета), calcNode.cpp(содержат описание логики вычислительного узла), handlerNode.cpp (содержит описание логики управляющего узла).

Каждый вычислительный узел при создании получает номер порта родителя, к которому он должен подключиться, а также свой id. Внутри себя он содержит 3 хэш-таблицы, содержащие сокеты детей узла, их идентификаторы процессов и номера портов.

При получении нового сообщения, адресованного конкретному узлу, строится путь до этого узла - вектор, содержащий id узлов на пути от управляющего к указанному и сообщение посылается в основной сокет, откуда согласно полученному пути пересылается через другие сокеты к необходимому узлу.

3 Исходный код

socketRoutine.hpp

```
#pragma once

#include <zmq.hpp>
#include <unistd.h>
#include <string>

bool SendMessage(zmq::socket_t& socket, const std::string& message
    );

std::string RecieveMessage(zmq::socket_t& socket);

int BindSocket(zmq::socket_t& socket);

void CreateNode(int id, int portNumber);
```

socketRoutine.cpp

```
#include "socketRoutine.hpp"
#include <iostream>

bool SendMessage(zmq::socket_t& socket, const std::string& message
    ) {
    zmq::message_t m(message.size());
    memcpy(m.data(), message.c_str(), message.size());
    return socket.send(m);
}

std::string RecieveMessage(zmq::socket_t& socket) {
    zmq::message_t message;
    bool messageRecieved;
    try {
        messageRecieved = socket.recv(&message);
    } catch(...) {
        messageRecieved = false;
    }
    std::string recieved(static_cast<char*>(message.data()),
    message.size());

    if(!messageRecieved || recieved.empty()) {
        return "Error: Node is unavailable";
    } else {
        return recieved;
    }
}

int BindSocket(zmq::socket_t& socket) {
    int port = 50000;
    std::string portTemplate = "tcp://127.0.0.1:";
    while(true) {
        try {
            socket.bind(portTemplate + std::to_string(port));
            break;
        } catch(...) {
            port++;
        }
    }
    return port;
```

```
}

void CreateNode(int id, int portNumber) {
    char* arg0 = strdup("./calcNode");
    char* arg1 = strdup((std::to_string(id)).c_str());
    char* arg2 = strdup((std::to_string(portNumber)).c_str());
    char* args[] = {arg0, arg1, arg2, NULL};
    execv("./calcNode", args);
}
```

calcNode.cpp

```
#include <string>
#include <chrono>
#include <sstream>
#include <zmq.hpp>
#include <csignal>
#include <iostream>
#include <unordered_map>

#include "socketRoutine.hpp"

int main(int argc, char* argv[]) {
    if(argc != 3) {
        std::cerr << "Not enough parameters" << std::endl;
        exit(-1);
    }
    int id = std::stoi(argv[1]);
    int parentPort = std::stoi(argv[2]);
    zmq::context_t ctx;
    zmq::socket_t parentSocket(ctx, ZMQ_REP);
    std::string portTemplate = "tcp://127.0.0.1:";
    parentSocket.connect(portTemplate + std::to_string(parentPort)
    );
    std::unordered_map<int, zmq::socket_t> sockets;
    std::unordered_map<int, int> pids;
    std::unordered_map<int, int> ports;
    auto start = std::chrono::high_resolution_clock::now();
    auto stop = std::chrono::high_resolution_clock::now();
    auto time = 0;
    bool clockStarted = false;
    while(true) {
        std::string action = RecieveMessage(parentSocket);
        std::stringstream s(action);
        std::string command;
        s >> command;
        if(command == "pid") {
            std::string reply = "Ok: " + std::to_string(getpid());
            SendMessage(parentSocket, reply);
        } else if(command == "create") {
            int size, nodeId;
            s >> size;
            std::vector<int> path(size);
            for(int i = 0; i < size; ++i) {
                s >> path[i];
            }
            s >> nodeId;
            if(size == 0) {
                sockets.insert(std::make_pair(nodeId, zmq::
    socket_t(ctx, ZMQ_REQ)));
```

```
47                int port = BindSocket(sockets.at(nodeId));
48                int pid = fork();
49                if(pid == -1) {
50                    SendMessage(parentSocket, "Unable to fork");
51                } else if(pid == 0) {
52                    CreateNode(nodeId, port);
53                } else {
54                    ports[nodeId] = port;
55                    pids[nodeId] = pid;
56                    SendMessage(sockets.at(nodeId), "pid");
57                    SendMessage(parentSocket, RecieveMessage(
    sockets.at(nodeId)));
58                }
59            } else {
60                int nextId = path.front();
61                path.erase(path.begin());
62                std::stringstream msg;
63                msg << "create " << path.size();
64                for(int i : path) {
65                    msg << " " << i;
66                }
67                msg << " " << nodeId;
68                SendMessage(sockets.at(nextId), msg.str());
69                SendMessage(parentSocket, RecieveMessage(sockets.
    at(nextId)));
70            }
71        } else if(command == "remove") {
72            int size, nodeId;
73            s >> size;
74            std::vector<int> path(size);
75            for(int i = 0; i < size; ++i) {
76                s >> path[i];
77            }
78            s >> nodeId;
79            if(path.empty()) {
80                SendMessage(sockets.at(nodeId), "kill");
81                RecieveMessage(sockets.at(nodeId));
82                kill(pids[nodeId], SIGTERM);
83                kill(pids[nodeId], SIGKILL);
84                pids.erase(nodeId);
85                sockets.at(nodeId).disconnect(portTemplate + std::
    to_string(ports[nodeId]));
86                ports.erase(nodeId);
87                sockets.erase(nodeId);
88                SendMessage(parentSocket, "Ok");
89            } else {
90                int nextId = path.front();
91                path.erase(path.begin());
92                std::stringstream msg;
93                msg << "remove " << path.size();
94                for(int i : path) {
95                    msg << " " << i;
96                }
97                msg << " " << nodeId;
98                SendMessage(sockets.at(nextId), msg.str());
99                SendMessage(parentSocket, RecieveMessage(sockets.
    at(nextId)));
100           }
101       } else if(command == "exec") {
```

```
102             int size;
103             std::string subcommand;
104             s >> subcommand >> size;
105             std::vector<int> path(size);
106             for(int i = 0; i < size; ++i) {
107                 s >> path[i];
108             }
109             if(path.empty()) {
110                 if(subcommand == "start") {
111                     start = std::chrono::high_resolution_clock::
    now();
112                     clockStarted = true;
113                     SendMessage(parentSocket, "Ok:" + std::
    to_string(id));
114                 } else if(subcommand == "stop") {
115                     if(clockStarted) {
116                         stop = std::chrono::high_resolution_clock
    ::now();
117                         time += std::chrono::duration_cast<std::
    chrono::milliseconds>
118                                 (stop - start).count();
119                         clockStarted = false;
120                     }
121                     SendMessage(parentSocket, "Ok:" + std::
    to_string(id));
122                 } else if(subcommand == "time") {
123                     SendMessage(parentSocket, "Ok: " + std::
    to_string(id) + ": "
124                             + std::to_string(time));
125                 }
126             } else {
127                 int nextId = path.front();
128                 path.erase(path.begin());
129                 std::stringstream msg;
130                 msg << "exec " << subcommand << " " << path.size()
    ;
131                 for(int i : path) {
132                     msg << " " << i;
133                 }
134                 SendMessage(sockets.at(nextId), msg.str());
135                 SendMessage(parentSocket, RecieveMessage(sockets.
    at(nextId)));
136             }
137         } else if(command == "ping") {
138             int size;
139             s >> size;
140             std::vector<int> path(size);
141             for(int i = 0; i < size; ++i) {
142                 s >> path[i];
143             }
144             if(path.empty()) {
145                 SendMessage(parentSocket, "Ok: 1");
146             } else {
147                 int nextId = path.front();
148                 path.erase(path.begin());
149                 std::stringstream msg;
150                 msg << "ping " << path.size();
151                 for(int i : path) {
152                     msg << " " << i;
```

```
                }
                SendMessage(sockets.at(nextId), msg.str());
                SendMessage(parentSocket, RecieveMessage(sockets.
    at(nextId)));
            }
        } else if(command == "kill") {
            for(auto& item : sockets) {
                SendMessage(item.second, "kill");
                RecieveMessage(item.second);
                kill(pids[item.first], SIGTERM);
                kill(pids[item.first], SIGKILL);
            }
            SendMessage(parentSocket, "Ok");
        }
        if(parentPort == 0) {
            break;
        }
    }
}
```

handlerNode.cpp

```
#include <iostream>
#include <chrono>
#include <string>
#include <zmq.hpp>
#include <vector>
#include <csignal>
#include <sstream>
#include <memory>
#include <unordered_map>

#include "socketRoutine.hpp"

struct TreeNode {
    TreeNode(int id, std::weak_ptr<TreeNode> parent) : id(id),
    parent(parent) {};
    int id;
    std::weak_ptr<TreeNode> parent;
    std::unordered_map<int, std::shared_ptr<TreeNode>> children;
};

class NTree {
public:
    bool Insert(int nodeId, int parentId) {
        if(root == nullptr) {
            root = std::make_shared<TreeNode>(nodeId, std::
    weak_ptr<TreeNode>());
            return true;
        }
        std::vector<int> pathToNode = PathTo(parentId);
        if(pathToNode.empty()) {
            return false;
        }
        pathToNode.erase(pathToNode.begin());
        std::shared_ptr<TreeNode> temp = root;
        for(const auto& node : pathToNode) {
            temp = temp->children[node];
        }
        temp->children[nodeId] = std::make_shared<TreeNode>(nodeId
```

```
    , temp);
37          return true;
38      }
39
40      bool Remove(int nodeId) {
41          std::vector<int> pathToNode = PathTo(nodeId);
42          if(pathToNode.empty()) {
43              return false;
44          }
45          pathToNode.erase(pathToNode.begin());
46          std::shared_ptr<TreeNode> temp = root;
47          for(const auto& node : pathToNode) {
48              temp = temp->children[node];
49          }
50          if(temp->parent.lock()) {
51              temp = temp->parent.lock();
52              temp->children.erase(nodeId);
53          } else {
54              root = nullptr;
55          }
56          return true;
57      }
58      std::vector<int> PathTo(int id) const {
59          std::vector<int> path;
60          if(!findNode(root, id, path)) {
61              return {};
62          } else {
63              return path;
64          }
65      }
66  private:
67      bool findNode(std::shared_ptr<TreeNode> current, int id, std::
    vector<int>& path) const {
68          if(!current) {
69              return false;
70          }
71          if(current->id == id) {
72              path.push_back(current->id);
73              return true;
74          }
75          path.push_back(current->id);
76          for(const auto& node : current->children) {
77              if(findNode(node.second, id, path)) {
78                  return true;
79              }
80          }
81          path.pop_back();
82          return false;
83      }
84      std::shared_ptr<TreeNode> root = nullptr;
85  };
86
87  int main() {
88      NTree calcs;
89      std::string action;
90      int childPid = 0;
91      int childId = 0;
92      zmq::context_t ctx(1);
93      zmq::socket_t handlerSocket(ctx, ZMQ_REQ);
```

```
94      handlerSocket.setsockopt(ZMQ_SNDTIMEO, 2000);
95      handlerSocket.setsockopt(ZMQ_LINGER, 0);
96      int portNumber = BindSocket(handlerSocket);
97      while(true) {
98          std::cin >> action;
99          if(action == "create") {
100             int nodeId, parentId;
101             std::string result;
102             std::cin >> nodeId >> parentId;
103             if(!childPid) {
104                 childPid = fork();
105                 if(childPid == -1) {
106                     std::cout << "Unable to create process" << std
    ::endl;
107                     exit(-1);
108                 } else if(childPid == 0) {
109                     CreateNode(nodeId, portNumber);
110                 } else {
111                     parentId = 0;
112                     childId = nodeId;
113                     SendMessage(handlerSocket, "pid");
114                     result = RecieveMessage(handlerSocket);
115                 }
116             } else {
117                 if(!calcs.PathTo(nodeId).empty()) {
118                     std::cout << "Error: Already exists" << std::
    endl;
119                     continue;
120                 }
121                 std::vector<int> path = calcs.PathTo(parentId);
122                 if(path.empty()) {
123                     std::cout << "Error: Parent not found" << std
    ::endl;
124                     continue;
125                 }
126                 path.erase(path.begin());
127                 std::stringstream s;
128                 s << "create " << path.size();
129                 for(int id : path) {
130                     s << " " << id;
131                 }
132                 s << " " << nodeId;
133                 SendMessage(handlerSocket, s.str());
134                 result = RecieveMessage(handlerSocket);
135             }
136
137             if(result.substr(0, 2) == "Ok") {
138                 calcs.Insert(nodeId, parentId);
139             }
140             std::cout << result << std::endl;
141         } else if(action == "remove") {
142             if(childPid == 0) {
143                 std::cout << "Error: Not found" << std::endl;
144                 continue;
145             }
146             int nodeId;
147             std::cin >> nodeId;
148             if(nodeId == childId) {
149                 SendMessage(handlerSocket, "kill");
```

```
                RecieveMessage(handlerSocket);
                kill(childPid, SIGTERM);
                kill(childPid, SIGKILL);
                childId = 0;
                childPid = 0;
                std::cout << "Ok" << std::endl;
                calcs.Remove(nodeId);
                continue;
            }
            std::vector<int> path = calcs.PathTo(nodeId);
            if(path.empty()) {
                std::cout << "Error: Not found" << std::endl;
                continue;
            }
            path.erase(path.begin());
            std::stringstream s;
            s << "remove " << path.size() - 1;
            for(int i : path) {
                s << " " << i;
            }
            SendMessage(handlerSocket, s.str());
            std::string recieved = RecieveMessage(handlerSocket);
            if(recieved.substr(0, 2) == "Ok") {
                calcs.Remove(nodeId);
            }
            std::cout << recieved << std::endl;
        } else if(action == "exec") {
            int nodeId;
            std::string subcommand;
            std::cin >> nodeId >> subcommand;
            std::vector<int> path = calcs.PathTo(nodeId);
            if(path.empty()) {
                std::cout << "Error: Not found" << std::endl;
                continue;
            }
            path.erase(path.begin());
            std::stringstream s;
            s << "exec " << subcommand << " " << path.size();
            for(int i : path) {
                s << " " << i;
            }
            std::cout << std::endl;
            SendMessage(handlerSocket, s.str());
            std::string recieved = RecieveMessage(handlerSocket);
            std::cout << recieved << std::endl;
        } else if(action == "ping") {
            if(childPid == 0) {
                std::cout << "Error: Not found" << std::endl;
                continue;
            }
            int nodeId;
            std::cin >> nodeId;
            std::vector<int> path = calcs.PathTo(nodeId);
            if(path.empty()) {
                std::cout << "Error: Not found" << std::endl;
                continue;
            }
            path.erase(path.begin());
            std::stringstream s;
```

```
209             s << "ping " << path.size();
210             for(int i : path) {
211                 s << " " << i;
212             }
213             SendMessage(handlerSocket, s.str());
214             std::string recieved = RecieveMessage(handlerSocket);
215             std::cout << recieved << std::endl;
216         } else if(action == "exit") {
217             SendMessage(handlerSocket, "kill");
218             RecieveMessage(handlerSocket);
219             kill(childPid, SIGTERM);
220             kill(childPid, SIGKILL);
221             break;
222         } else {
223             std::cout << "Unknown command" << std::endl;
224         }
225     }
226     return 0;
227 }
```

4 Консоль

```
qelderdelta@qelderdelta-UX331UA:~/Study/OS/lab6/build$ ./main
create 2 3
Ok: 13276
create 3 2
Ok: 13279
create 4 2
Ok: 13282
create 5 3
Ok: 13285
ping 5
Ok: 1
ping 6
Error: Not found
ping 2
Ok: 1
exec 4 time
Ok: 4: 0
exec 4 start
Ok:4
exec 4 stop
Ok:4
exec 4 time
Ok: 4: 10909
exit
qelderdelta@qelderdelta-UX331UA:~/Study/OS/lab6/build$ ps
PID TTY          TIME CMD
12523 pts/0    00:00:00 bash
13290 pts/0    00:00:00 ps
```

5 Выводы

В ходе выполнения лабораторной работы я познакомился с очередью сообщений ZeroMQ, а конкретнее с сокетами, реализующими паттерн Request-Reply, а также получил опыт написания распределенных систем.