Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №5-7 по курсу**

**«Операционные системы»**

Студент: Ибрагимов Далгат Магмоедалиевич

Группа: М8О-208Б-22

Вариант: 28

Преподаватель: Миронов Евгений Сергеевич

Оценка: \_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2023

**Содержание**

1. Репозиторий

2. Постановка задачи

3. Общие сведения о программе

4. Задание

5. Исходный код

6. Демонстрация работы программы

7. Выводы

**Репозиторий**

<https://github.com/L0ckR/OS_LABS>

**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.

**Постановка задачи**

Топология 2:

Все вычислительные узлы находятся в дереве общего вида. Есть только один управляющий узел. Чтобы добавить новый вычислительный узел к управляющему, то необходимо выполнить команду: create id -1.

Набор команд 1:

Подсчет суммы n чисел - формат команды: exec id n k\_1 k\_2 k\_n

id – целочисленный идентификатор вычислительного узла, на который отправляется команда n – количество складываемых чисел, k\_1 … k\_n– складываемые числа.

Команда проверки 2:

Формат команды: ping id Команда проверяет доступность конкретного узла. Если узла нет, то необходимо выводить ошибку: «Error: Not found».

В ходе выполнения лабораторной работы я использовала библиотеку ZeroMQ и следующие команды:

* bind() - устанавливает "сокет" на адрес, а затем принимает входящие соединения на этом адресе.
* unbind() - отвязывает сокет от адреса
* connect() - создание соединения между сокетом и адресом
* disconnect() - разрывает соединение между сокетом и адресом
* send() - отправка сообщений
* recv() - получение сообщений

**Общие сведения о программе**

Программа написана для операционной системы Linux. Для реализации очереди сообщений была использована библиотека ZeroMQ. Были написаны две программы – для управляющего узла и для вычислительного узла.

**Исходный код**

**========================== node\_routine.hpp ==========================**

**#pragma once**

**#include <iostream>**

**#include <sstream>**

**#include <unordered\_map>**

**#include <optional>**

**#include <memory>**

**#include "unistd.h"**

**#include "socket\_routine.hpp"**

**class Node{**

**private:**

**zmq::context\_t context;**

**public:**

**std::unordered\_map<int, std::unique\_ptr<zmq::socket\_t> > children;**

**std::unordered\_map<int, int> childrenPort;**

**int id;**

**zmq::socket\_t parent;**

**int parentPort;**

**Node(int \_id, int \_parentPort = -1): id(\_id), parent(context, ZMQ\_REP), parentPort(\_parentPort) {**

**if(\_id != -1) {**

**Connect(&parent, \_parentPort);**

**}**

**}**

**std::string Ping(int \_id);**

**std::string Create(int idChild, const std::string& programPath);**

**std::string Pid();**

**std::string Send(const std::string& str, int \_id);**

**std::string Kill();**

**};**

**======================== socket\_routine.hpp ========================**

**#pragma once**

**#include <iostream>**

**#include <sstream>**

**#include <string>**

**#include <optional>**

**#include <zmq.hpp>**

**int Bind(zmq::socket\_t \*socket, int id);**

**void Unbind(zmq::socket\_t \*socket, int port);**

**void Connect(zmq::socket\_t \*socket, int port);**

**void Disconnect(zmq::socket\_t \*socket, int port);**

**bool SendMessage(zmq::socket\_t \*socket, const std::string& msg);**

**std::optional<std::string> ReceiveMessage(zmq::socket\_t \*socket);**

**========================= node\_routine.cpp ==========================**

**#include "node\_routine.hpp"**

**std::string Node::Ping(int \_id) {**

**std::string ans = "Ok: 0";**

**if (\_id == id) {**

**ans = "Ok: 1";**

**return ans;**

**} else if (auto it = children.find(\_id); it != children.end()) {**

**std::string msg = "ping " + std::to\_string(\_id);**

**SendMessage(it->second.get(), msg);**

**if (auto msg = ReceiveMessage(children[\_id].get()); msg.has\_value(), msg == "Ok: 1") {**

**ans = \*msg;**

**}**

**return ans;**

**}**

**return ans;**

**}**

**std::string Node::Create(int idChild, const std::string& programPath) {**

**std::string programName = programPath.substr(programPath.find\_last\_of("/") + 1);**

**children[idChild] = std::make\_unique<zmq::socket\_t>(context, ZMQ\_REQ);**

**int newPort = Bind(children[idChild].get(), idChild);**

**childrenPort[idChild] = newPort;**

**int pid = fork();**

**if (pid == 0) { // Child process**

**execl(programPath.c\_str(), programName.c\_str(), std::to\_string(idChild).c\_str(), std::to\_string(newPort).c\_str(), nullptr);**

**} else { // Parent process**

**std::string pidChild = "Error: couldn't connect to child";**

**children[idChild]->setsockopt(ZMQ\_SNDTIMEO, 3000);**

**SendMessage(children[idChild].get(), "pid");**

**if (auto msg = ReceiveMessage(children[idChild].get()); msg.has\_value()) {**

**pidChild = \*msg;**

**}**

**return "Ok: " + pidChild;**

**}**

**return 0;**

**}**

**std::string Node::Pid() {**

**return std::to\_string(getpid());**

**}**

**std::string Node::Send(const std::string& str, int id) {**

**if (children.size() == 0) {**

**return "Error: Not found";**

**} else if (auto it = children.find(id); it != children.end()) {**

**if (SendMessage(it->second.get(), str)) {**

**std::string ans = "Error: Not found";**

**if (auto msg = ReceiveMessage(children[id].get()); msg.has\_value()) {**

**ans = \*msg;**

**}**

**return ans;**

**}**

**} else {**

**std::string ans = "Error: Not found";**

**for (auto& child : children) {**

**std::string msg = "send " + std::to\_string(id) + " " + str;**

**if (SendMessage(child.second.get(), msg)) {**

**if (auto msg = ReceiveMessage(child.second.get()); msg.has\_value()) {**

**ans = \*msg;**

**}**

**}**

**}**

**return ans;**

**}**

**return 0;**

**}**

**std::string Node::Kill() {**

**std::string ans;**

**for (auto& child : children) {**

**std::string msg = "kill";**

**if (SendMessage(child.second.get(), msg)) {**

**if (auto tmp = ReceiveMessage(child.second.get()); tmp.has\_value()) {**

**msg = \*tmp;**

**}**

**if (ans.size() > 0) {**

**ans = ans + " " + msg;**

**} else {**

**ans = msg;**

**}**

**}**

**Unbind(child.second.get(), childrenPort[child.first]);**

**child.second->close();**

**}**

**children.clear();**

**childrenPort.clear();**

**return ans;**

**}**

**======================== socket\_routine.cpp ========================**

**#include "socket\_routine.hpp"**

**int Bind(zmq::socket\_t \*socket, int id) {**

**int port = 4040 + id;**

**while(true) {**

**std::string address = "tcp://127.0.0.1:" + std::to\_string(port);**

**try{**

**socket->bind(address);**

**break;**

**} catch(...) {**

**port++;**

**}**

**}**

**return port;**

**}**

**void Unbind(zmq::socket\_t \*socket, int port) {**

**std::string address = "tcp://127.0.0.1:" + std::to\_string(port);**

**socket->unbind(address);**

**}**

**void Connect(zmq::socket\_t \*socket, int port) {**

**std::string address = "tcp://127.0.0.1:" + std::to\_string(port);**

**socket->connect(address);**

**}**

**void Disconnect(zmq::socket\_t \*socket, int port) {**

**std::string address = "tcp://127.0.0.1:" + std::to\_string(port);**

**socket->disconnect(address);**

**}**

**bool SendMessage(zmq::socket\_t \*socket, const std::string& msg) {**

**zmq::message\_t message(msg.size());**

**memcpy(message.data(), msg.c\_str(), msg.size());**

**try {**

**socket->send(message, 0);**

**return true;**

**} catch(...) {**

**return false;**

**}**

**}**

**std::optional<std::string> ReceiveMessage(zmq::socket\_t \*socket) {**

**zmq::message\_t message;**

**socket->recv(&message, 0);**

**std::string received(static\_cast<char\*>(message.data()), message.size());**

**return received.empty() ? std::nullopt : std::make\_optional(received);**

**}**

**========================== server.cpp ==========================**

**#include "node\_routine.hpp"**

**#include "socket\_routine.hpp"**

**#include <fstream>**

**#include <signal.h>**

**int main(int argc, char \*\*argv) {**

**if (argc != 3) {**

**perror("Not enough arguments");**

**exit(EXIT\_FAILURE);**

**}**

**Node task(atoi(argv[1]), atoi(argv[2]));**

**std::string programPath = getenv("PROGRAM\_PATH");**

**while(1) {**

**std::string message;**

**std::string command = " ";**

**if (auto msg = ReceiveMessage(&(task.parent)); msg.has\_value()) {**

**message = \*msg;**

**}**

**std::istringstream request(message);**

**request >> command;**

**if (command == "create") {**

**int idChild;**

**request >> idChild;**

**std::string ans = task.Create(idChild, programPath);**

**SendMessage(&task.parent, ans);**

**} else if (command == "pid") {**

**std::string ans = task.Pid();**

**SendMessage(&task.parent, ans);**

**} else if (command == "ping") {**

**int idChild;**

**request >> idChild;**

**std::string ans = task.Ping(idChild);**

**SendMessage(&task.parent, ans);**

**} else if (command == "send") {**

**int id;**

**request >> id;**

**std::string str;**

**getline(request, str);**

**str.erase(0, 1);**

**std::string ans;**

**ans = task.Send(str, id);**

**SendMessage(&task.parent, ans);**

**} else if (command == "exec") {**

**int cnt, sum = 0, number;**

**request >> cnt;**

**for(int i = 0; i < cnt; i++) {**

**request >> number;**

**sum += number;**

**}**

**std::string to\_send;**

**to\_send = "Ok: " + std::to\_string(task.id) + ": " + std::to\_string(sum);**

**SendMessage(&task.parent,to\_send);**

**} else if (command == "kill") {**

**std::string ans = task.Kill();**

**ans = std::to\_string(task.id) + " " + ans;**

**SendMessage(&task.parent, ans);**

**Disconnect(&task.parent, task.parentPort);**

**task.parent.close();**

**break;**

**}**

**}**

**return 0;**

**}**

**========================== client.cpp ==========================**

**#include "set"**

**#include "node\_routine.hpp"**

**#include "socket\_routine.hpp"**

**int main() {**

**std::set<int> Nodes;**

**std::string programPath = getenv("PROGRAM\_PATH");**

**Node task(-1);**

**Nodes.insert(-1);**

**std::string command;**

**while (std::cin >> command) {**

**if (command == "create") {**

**int idChild, idParent;**

**std::cin >> idChild >> idParent;**

**if (Nodes.find(idChild) != Nodes.end()) {**

**std::cout << "Error: Already exists" << std::endl;**

**} else if (Nodes.find(idParent) == Nodes.end()) {**

**std::cout << "Error: Parent not found" << std::endl;**

**}else if (idParent == task.id) { // from -1**

**std::string ans = task.Create(idChild, programPath);**

**std::cout << ans << std::endl;**

**Nodes.insert(idChild);**

**} else { // from other node**

**std::string str = "create " + std::to\_string(idChild);**

**std::string ans = task.Send(str, idParent);**

**std::cout << ans << std::endl;**

**Nodes.insert(idChild);**

**}**

**} else if (command == "ping") {**

**int idChild;**

**std::cin >> idChild;**

**if (Nodes.find(idChild) == Nodes.end()) {**

**std::cout << "Error: Not found" << std::endl;**

**} else if (task.children.find(idChild) != task.children.end()) {**

**std::string ans = task.Ping(idChild);**

**std::cout << ans << std::endl;**

**} else {**

**std::string str = "ping " + std::to\_string(idChild);**

**std::string ans = task.Send(str, idChild);**

**if (ans == "Error: Not found") {**

**ans = "Ok: 0";**

**}**

**std::cout << ans << std::endl;**

**}**

**}else if (command == "exec") {**

**int id, number, count;**

**std::cin >> id >> count;**

**std::string msg = "exec " + std::to\_string(count);**

**for (int i = 0; i < count; i++) {**

**std::cin >> number;**

**msg += " " + std::to\_string(number);**

**}**

**if (Nodes.find(id) == Nodes.end()) {**

**std::cout << "Error: Not found" << std::endl;**

**} else {**

**std::string ans = task.Send(msg, id);**

**std::cout << ans << std::endl;**

**}**

**}else if(command == "kill") {**

**int id;**

**std::cin >> id;**

**std::string msg = "kill";**

**if (Nodes.find(id) == Nodes.end()) {**

**std::cout << "Error: Not found" << std::endl;**

**} else {**

**std::string ans = task.Send(msg, id);**

**if (ans != "Error: Not found") {**

**std::istringstream ids(ans);**

**int tmp;**

**while(ids >> tmp) {**

**Nodes.erase(tmp);**

**}**

**ans = "Ok";**

**if(task.children.find(id) != task.children.end()) {**

**Unbind(task.children[id].get(), task.childrenPort[id]);**

**task.children[id]->close();**

**task.children.erase(id);**

**task.childrenPort.erase(id);**

**}**

**}**

**std::cout << ans << std::endl;**

**}**

**} else if (command == "exit") {**

**task.Kill();**

**return 0;**

**}**

**}**

**}**

**Демонстрация работы программы**



**Выводы**

В ходе выполнения лабораторной работе я получил знания и навыки использования очередей сообщений. Я получил понимание концепции асинхронного программирования, узнал о сокетах и сетевом протоколе TCP.