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»**

**УПРАВЛЕНИЕ СЕРВЕРАМИ СООБЩЕНИЙ, ПРИМЕНЕНИЕ ОТЛОЖЕННЫХ ВЫЧИСЛЕНИЙ, ИНТЕГРАЦИЯ ПРОГРАММНЫХ СИСТЕМ ДРУГ С ДРУГОМ**

Студент: Салихов Тимур Русланович

Группа: М8О–206Б–20

Вариант: 45

Преподаватель: Соколов Андрей Алексеевич

Оценка: \_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_

Москва, 2020.

**Постановка задачи**

## Цель работы

## Целью является приобретение практических навыков в:

## •  Управлении серверами сообщений (No5)

## •  Применение отложенных вычислений (No6)

## •  Интеграция программных систем друг с другом (No7)

## 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. 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. Список основных поддерживаемых команд:

**Создание нового вычислительного узла**

Формат команды: create id [parent]  
id – целочисленный идентификатор нового вычислительного узла

parent – целочисленный идентификатор родительского узла. Если топологией не предусмотрено введение данного параметра, то его необходимо игнорировать (если его ввели)

Формат вывода:

«Ok: pid», где pid – идентификатор процесса для созданного вычислительного узла

«Error: Already exists» - вычислительный узел с таким идентификатором уже существует

«Error: Parent not found» - нет такого родительского узла с таким идентификатором

«Error: Parent is unavailable» - родительский узел существует, но по каким-то причинам с ним не удается связаться

«Error: [Custom error]» - любая другая обрабатываемая ошибка Пример:  
> create 10 5  
Ok: 3128

*Примечания: создание нового управляющего узла осуществляется пользователем программы при помощи запуска исполняемого файла. Id и pid — это разные идентификаторы.*

**Исполнение команды на вычислительном узле**

Формат команды: exec id [params]

id – целочисленный идентификатор вычислительного узла, на который отправляется команда

Формат вывода:

«Ok:id: [result]», где result – результат выполненной команды

«Error:id: Not found» - вычислительный узел с таким идентификатором не найден

«Error:id: Node is unavailable» - по каким-то причинам не удается связаться с вычислительным узлом

«Error:id: [Custom error]» - любая другая обрабатываемая ошибка  
Пример:  
Можно найти в описании конкретной команды, определенной вариантом задания.

*Примечание: выполнение команд должно быть асинхронным. Т.е. пока выполняется команда на одном из вычислительных узлов, то можно отправить следующую команду на другой вычислительный узел.*

**Топология 4**

Аналогично топологии 4, но узлы находятся в идеально сбалансированном бинарном дереве. Каждый следующий узел должен добавляться в самое наименьшее поддерево.

**Набора команд 2 (локальный целочисленный словарь)**Формат команды сохранения значения: exec id name value  
id – целочисленный идентификатор вычислительного узла, на который отправляется команда name – ключ, по которому будет сохранено значение (строка формата [A-Za-z0-9]+)  
value – целочисленное значение  
Формат команды загрузки значения: exec id name  
Пример:  
> exec 10 MyVar  
Ok:10: 'MyVar' not found

> exec 10 MyVar 5  
Ok:10  
> exec 12 MyVar  
Ok:12: 'MyVar' not found > exec 10 MyVar

Ok:10: 5  
> exec 10 MyVar 7  
Ok:10  
> exec 10 MyVar  
Ok:10: 7  
*Примечания: Можно использовать std:map.*

**Команда проверки 2**Формат команды: ping id

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

Пример:  
> ping 10  
Ok: 1 // узел 10 доступен  
> ping 17  
Ok: 0 // узел 17 недоступен

**Общие сведения о программе**

Программа компилируется из файла server.cpp. Также используется библиотека ZeroMQ.

**Общий метод и алгоритм решения**.

Для реализации поставленной задачи необходимо:

1. Изучить библиотеку ZeroMQ.
2. Написать сбалансированное бинарное дерево
3. Написать функции с использованием ZMQ
4. Реализовать create, exec, ping, kill

**Основные файлы программы**

**balancedTree.cpp:**

#pragma once

#include <algorithm>

#include <set>

class BalancedTree {

class BalancedTreeNode {

public:

int id;

BalancedTreeNode\* left;

BalancedTreeNode\* right;

int height;

bool available;

BalancedTreeNode(int id) {

this->id = id;

available = true;

left = NULL;

right = NULL;

}

void CheckAvailability(int id) {

if (this->id == id) {

available = false;

} else {

if (left != NULL) {

left->CheckAvailability(id);

}

if (right != NULL) {

right->CheckAvailability(id);

}

}

}

void Remove(int id, std::set<int>& ids) {

if (left != NULL && left->id == id) {

left->RecursionRemove(ids);

ids.erase(left->id);

delete left;

left = NULL;

} else if (right != NULL && right->id == id) {

right->RecursionRemove(ids);

ids.erase(right->id);

delete right;

right = NULL;

} else {

if (left != NULL) {

left->Remove(id, ids);

}

if (right != NULL) {

right->Remove(id, ids);

}

}

}

void RecursionRemove(std::set<int>& ids) {

if (left != NULL) {

left->RecursionRemove(ids);

ids.erase(left->id);

delete left;

left = NULL;

}

if (right != NULL) {

right->RecursionRemove(ids);

ids.erase(right->id);

delete right;

right = NULL;

}

}

void AddInNode(int id, int parent\_id, std::set<int>& ids) {

if (this->id == parent\_id) {

if (left == NULL) {

left = new BalancedTreeNode(id);

} else {

right = new BalancedTreeNode(id);

}

ids.insert(id);

} else {

if (left != NULL) {

left->AddInNode(id, parent\_id, ids);

}

if (right != nullptr) {

right->AddInNode(id, parent\_id, ids);

}

}

}

int MinimalHeight() {

if (left == NULL || right == NULL) {

return 0;

}

int left\_height = -1;

int right\_height = -1;

if (left != NULL && left->available == true) {

left\_height = left->MinimalHeight();

}

if (right != NULL && right->available == true) {

right\_height = right->MinimalHeight();

}

if (right\_height == -1 && left\_height == -1) {

available = false;

return -1;

} else if (right\_height == -1) {

return left\_height + 1;

} else if (left\_height == -1) {

return right\_height + 1;

} else {

return std::min(left\_height, right\_height) + 1;

}

}

int IDMinimalHeight(int height, int current\_height) {

if (height < current\_height) {

return -2;

} else if (height > current\_height) {

int current\_id = -2;

if (left != NULL && left->available == true) {

current\_id = left->IDMinimalHeight(height, (current\_height + 1));

}

if (right != NULL && right->available == true && current\_id == -2) {

current\_id = right->IDMinimalHeight(height, (current\_height + 1));

}

return current\_id;

} else {

if (left == NULL || right == NULL) {

return id;

}

return -2;

}

}

~BalancedTreeNode() {}

};

private:

BalancedTreeNode\* root;

public:

std::set<int> ids;

BalancedTree() {

root = new BalancedTreeNode(-1);

}

bool Exist(int id) {

if (ids.find(id) != ids.end()) {

return true;

}

return false;

}

void AvailabilityCheck(int id) {

root->CheckAvailability(id);

}

int FindID() {

int h = root->MinimalHeight();

return root->IDMinimalHeight(h, 0);

}

void AddInTree(int id, int parent) {

root->AddInNode(id, parent, ids);

}

void RemoveFromRoot(int idElem) {

root->Remove(idElem, ids);

}

~BalancedTree() {

root->RecursionRemove(ids);

delete root;

}

};

**client.cpp:**

#include "balancedTree.hpp"

#include "dictionaryNode.hpp"

#include "ZMQ.hpp"

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

DictionaryNode node(atoi(argv[1]), atoi(argv[2]), atoi(argv[3]));

while (true) {

std::string message;

std::string command;

message = receive\_message(node.parent);

std::istringstream request(message);

request >> command;

if (command == "pid") {

std::string answer = std::to\_string(getpid());

send\_message(node.parent, answer);

}

else if (command == "ping") {

int child;

request >> child;

std::string answer = node.ping(child);

send\_message(node.parent, answer);

}

else if (command == "create") {

int child;

request >> child;

std::string answer = node.create(child);

send\_message(node.parent, answer);

}

else if (command == "exec") {

std::string str;

getline(request, str);

std::string answer = node.exec(str);

send\_message(node.parent, answer);

}

else if (command == "kill") {

std::string answer = node.kill();

send\_message(node.parent, answer);

disconnect(node.parent, node.parent\_port);

node.parent.close();

break;

}

}

return 0**;**

}

**dictionaryNode.cpp:**

#pragma once

#include <unistd.h>

#include <iostream>

#include <map>

#include "ZMQ.hpp"

class DictionaryNode {

private:

zmq::context\_t context;

std::map<std::string, int> \_dictionary = {

{"one", 1},

{"two", 2},

{"twenty three", 23},

{"one hundred and four", 104}};

public:

zmq::socket\_t left, right, parent;

int left\_port, right\_port, parent\_port;

int id, left\_id = -2, right\_id = -2, parent\_id;

DictionaryNode(int id, int parent\_port, int parent\_id)

: id(id),

parent\_port(parent\_port),

parent\_id(parent\_id),

left(context, ZMQ\_REQ),

right(context, ZMQ\_REQ),

parent(context, ZMQ\_REP) {

if (id != -1) {

connect(parent, parent\_port);

}

}

std::string create(int child\_id) {

int port;

bool isleft = false;

if (left\_id == -2) {

left\_port = bind(left, child\_id);

left\_id = child\_id;

port = left\_port;

isleft = true;

}

else if (right\_id == -2) {

right\_port = bind(right, child\_id);

right\_id = child\_id;

port = right\_port;

}

else {

std::string fail = "Error: can not create the dictionary node";

return fail;

}

int fork\_id = fork();

if (fork\_id == 0) {

if (execl("./client", "client", std::to\_string(child\_id).c\_str(), std::to\_string(port).c\_str(), std::to\_string(id).c\_str(), NULL) == -1) {

std::cout << "Error: can not run the execl-command" << std::endl;

exit(EXIT\_FAILURE);

}

}

else {

std::string child\_pid;

try {

if (isleft) {

left.setsockopt(ZMQ\_SNDTIMEO, 3000);

send\_message(left, "pid");

child\_pid = receive\_message(left);

} else {

right.setsockopt(ZMQ\_SNDTIMEO, 3000);

send\_message(right, "pid");

child\_pid = receive\_message(right);

}

return "Ok: " + child\_pid;

} catch (int) {

std::string fail = "Error: can not connect to the child";

return fail;

}

}

}

std::string ping(int id) {

std::string answer = "Ok: 0";

if (this->id == id) {

answer = "Ok: 1";

return answer;

}

else if (left\_id == id) {

std::string message = "ping " + std::to\_string(id);

send\_message(left, message);

try {

message = receive\_message(left);

if (message == "Ok: 1") {

answer = message;

}

} catch (int) {

}

}

else if (right\_id == id) {

std::string message = "ping " + std::to\_string(id);

send\_message(right, message);

try {

message = receive\_message(right);

if (message == "Ok: 1") {

answer = message;

}

} catch (int) {

}

}

return answer;

}

std::string sendstring(std::string string, int id) {

std::string answer = "Error: Parent not found";

if (left\_id == -2 && right\_id == -2) {

return answer;

}

else if (left\_id == id) {

if (ping(left\_id) == "Ok: 1") {

send\_message(left, string);

try {

answer = receive\_message(left);

} catch (int) {

}

}

}

else if (right\_id == id) {

if (ping(right\_id) == "Ok: 1") {

send\_message(right, string);

try {

answer = receive\_message(right);

} catch (int) {

}

}

}

else {

if (ping(left\_id) == "Ok: 1") {

std::string message = "send " + std::to\_string(id) + " " + string;

send\_message(left, message);

try {

message = receive\_message(left);

} catch (int) {

message = "Error: Parent not found";

}

if (message != "Error: Parent not found") {

answer = message;

}

}

if (ping(right\_id) == "Ok: 1") {

std::string message = "send " + std::to\_string(id) + " " + string;

send\_message(right, message);

try {

message = receive\_message(right);

} catch (int) {

message = "Error: Parent not found";

}

if (message != "Error: Parent not found") {

answer = message;

}

}

}

return answer;

}

std::string exec(std::string string) {

std::istringstream string\_thread(string);

std::string key;

string\_thread >> key;

int result = \_dictionary[key];

std::string answer = "Ok: " + std::to\_string(id) + ": " + std::to\_string(result);

return answer;

}

std::string kill() {

if (left\_id != -2) {

if (ping(left\_id) == "Ok: 1") {

std::string message = "kill";

send\_message(left, message);

try {

message = receive\_message(left);

} catch (int) {

}

unbind(left, left\_port);

left.close();

}

}

if (right\_id != -2) {

if (ping(right\_id) == "Ok: 1") {

std::string message = "kill";

send\_message(right, message);

try {

message = receive\_message(right);

} catch (int) {

}

unbind(right, right\_port);

right.close();

}

}

return std::to\_string(parent\_id);

}

~DictionaryNode() {}

};

**server.cpp:**

#include <iostream>

#include "ZMQ.hpp"

#include "balancedTree.hpp"

#include "dictionaryNode.hpp"

void menu() {

std::cout << "---------------------\n";

std::cout << "Avaliable commands:\n";

std::cout << "1. create <id>\n";

std::cout << "2. exec <key>\n";

std::cout << "3. ping <id>\n";

std::cout << "4. kill <id>\n";

std::cout << "5. exit\n";

std::cout << "---------------------\n";

}

int main() {

std::string command;

DictionaryNode node(-1, -1, -1);

std::string answer;

BalancedTree tree;

menu();

while (std::cin >> command) {

int id;

if (command == "create") {

std::cin >> id;

if (tree.Exist(id)) {

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

} else {

while (true) {

int idParent = tree.FindID();

if (idParent == node.id) {

answer = node.create(id);

tree.AddInTree(id, idParent);

break;

} else {

std::string message = "create " + std::to\_string(id);

answer = node.sendstring(message, idParent);

if (answer == "Error: Parent not found") {

tree.AvailabilityCheck(idParent);

} else {

tree.AddInTree(id, idParent);

break;

}

}

}

std::cout << answer << std::endl;

}

}

else if (command == "exec") {

std::cin >> id;

if (!tree.Exist(id)) {

std::cout << "Error: Node does not exist!\n";

}

std::string str;

getline(std::cin, str);

if (!tree.Exist(id)) {

std::cout << "Error: Parent is not existed\n";

} else {

std::string message = "exec " + str;

answer = node.sendstring(message, id);

std::cout << answer << std::endl;

}

}

else if (command == "ping") {

std::cin >> id;

if (!tree.Exist(id)) {

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

} else if (node.left\_id == id || node.right\_id == id) {

answer = node.ping(id);

std::cout << answer << std::endl;

} else {

std::string message = "ping " + std::to\_string(id);

answer = node.sendstring(message, id);

std::cout << answer << std::endl;

}

}

else if (command == "kill") {

std::cin >> id;

std::string message = "kill";

if (!tree.Exist(id)) {

std::cout << "Error: Parent is not existed\n";

} else {

answer = node.sendstring(message, id);

if (answer != "Error: Parent not found") {

tree.RemoveFromRoot(id);

if (id == node.left\_id) {

node.left\_id = -2;

unbind(node.left, node.left\_port);

answer = "Ok";

} else if (id == node.right\_id) {

node.right\_id = -2;

unbind(node.right, node.right\_port);

answer = "Ok";

} else {

message = "clear " + std::to\_string(id);

answer = node.sendstring(message, stoi(answer));

}

std::cout << answer << std::endl;

}

}

}

else if (command == "exit") {

node.kill();

std::cout << "Exited successfully" << std::endl;

return 0;

}

else {

std::cout << "Please enter correct command!\n\n";

menu();

}

std::cout << "---------------------\n";

}

}

**ZMQ:**

#pragma once

#include <zmq.hpp>

const int MAIN\_PORT = 4040;

void send\_message(zmq::socket\_t& socket, const std::string& msg) {

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

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

socket.send(message);

}

std::string receive\_message(zmq::socket\_t& socket) {

zmq::message\_t message;

int chars\_read;

try {

chars\_read = (int)socket.recv(&message);

} catch (...) {

chars\_read = 0;

}

if (chars\_read == 0) {

throw -1;

}

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

return received\_msg;

}

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);

}

int bind(zmq::socket\_t& socket, int id) {

int port = MAIN\_PORT + id;

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

while (1) {

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);

}

**Пример работы**

timursalihov@MBP-Timur-2 ~/O/L/b/src (main) > /Users/timursalihov/OperatingSystems/LW5-7/build/src/server

---------------------

Avaliable commands:

1. create <id>

2. exec <key>

3. ping <id>

4. kill <id>

5. exit

---------------------

create 4

Ok: 16099

---------------------

ping 4

Ok: 1

---------------------

kill 4

Ok

---------------------

ping 4

Ok: 0

---------------------

create 10

Ok: 16103

---------------------

exec 10 one

Ok: 10: 1

---------------------

exec 10 two

Ok: 10: 2

---------------------

exec 10 twenty\_three

Ok: 10: 23

---------------------

exec 10 one\_hundred\_and\_four

Ok: 10: 104

---------------------

kill 10

Ok

---------------------

exit

Exited successfully

**Вывод**

В данной лабораторной работе реализована распределенная асинхронная система по обработке запросов. Для передачи данных использованы очереди сообщений ZeroMQ. Передача данных осуществлялась по протоколу TCP, общение между процессами в котором происходит через порты. В процессе подготовки к написанию работы, мне пришлось изучить работу ZeroMQ, а также самостоятельно установить библиотеку для работы с очередями сообщений.