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**

**“Очереди сообщений”**

Студент: Зубко Дмитрий Валерьевич

Группа: М8О-208Б-20

Вариант:35

Преподаватель: Миронов Евгений Сергеевич

Оценка: \_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2021

**Содержание**

1. Репозиторий
2. Постановка задачи
3. Общие сведения о программе
4. Общий метод и алгоритм решения
5. Исходный код
6. Демонстрация работы программы
7. Выводы

**Репозиторий**

[https://github.com/usernameMAI/OS/](https://github.com/usernameMAI/OS/tree/main/os_lab2)

**Постановка задачи**

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.  
  
Вариант 35. Команды:  
create id  
exec id n n1 n2… ni (набор чисел, требуется посчитать сумму)  
ping id

**Общие сведения о программе**

Программа содержит 6 файлов с кодом:

ServerProgram.cpp – программа сервера

ClientProgram.cpp – программа клиента

CalculationNode.h – программа, в которой реализованы основные команды

ZMQFunctions.h – файл с функциями zero MQ

BalancedTree.h – программа с идеально сбалансированным деревом

**Общий метод и алгоритм решения**

Программа содержит makefile:

g++ -fsanitize=address ClientProgram.cpp -lzmq -o client -w  
g++ -fsanitize=address ServerProgram.cpp -lzmq -o server –w

Создаётся две программы. Запускается client, который вызывает server от определенных значений client\_id, parent\_port и parent\_id. Когда клиент получает сообщение, то он отправляет его на сервер. Все операции проходят с объектом node, который принадлежит классу CalculationNode.

**Исходный код**

**ClientProgram.cpp**

#include <bits/stdc++.h>  
#include "CalculationNode.h"  
#include "ZMQFunctions.h"  
#include "BalancedTree.h"  
  
using namespace std;  
  
/\*  
  
 ВАРИАНТ 35  
 4 1 2  
 все вычислительные узлы хранятся в идеально сбалансированном бинарном дереве,  
 каждый следующий узел должен добавляться в самое наименьшее поддерево.  
  
 тип команды -- подсчет суммы n чисел  
 exec id n k1 k2 ... kn  
 id -- целочисленный идентификатор вычислительного узла, на который отправляется команда  
 n -- количество складываемых чисел  
 k1 ... kn -- складываемые числа  
  
 формат команды ping id  
 команда проверяет доступность конкретного узла, если узла нет, то необходимо  
 выводить ошибку.  
  
\*/  
  
int main() {  
  
 string command, ans;  
 BalancedTree tree;  
 CalculationNode node(-1, -1, -1);  
  
 cout << "Enter the command: \n\n";  
 cout << "create id: for creating a new calculation node\n";  
 cout << "exec id n n1 n2... n: for calculating a sum\n";  
 cout << "ping id: for checking node-availabilty\n";  
 cout << "kill id: for killing a calculation node\n\n";  
  
 while ((cout << "Enter command: ") && (cin >> command)) {  
  
 if (command == "create") {  
 int child;  
 cin >> child;  
 if (tree.Exist(child)) {  
 cout << "Error: Already exists" << std::endl;  
 } else {  
 while (true) {  
 int idParent = tree.FindID();  
 if (idParent == node.id) {  
 ans = node.create(child);  
 tree.AddInTree(child, idParent);  
 break;  
 } else {  
 string message = "create " + to\_string(child);  
 ans = node.sendstring(message, idParent);  
 if (ans == "Error: Parent not found") {  
 tree.AvailabilityCheck(idParent);  
 } else {  
 tree.AddInTree(child, idParent);  
 break;  
 }  
 }  
 }  
 cout << ans << endl;  
 }  
 } else if (command == "exec") {  
 string str;  
 int child;  
 cin >> child;  
 getline(cin, str);  
 if (!tree.Exist(child)) {  
 cout << "Error: Parent is not existed" << std::endl;  
 } else {  
 string message = "exec " + str;  
 ans = node.sendstring(message, child);  
 cout << ans << endl;  
 }  
 } else if (command == "ping") {  
 int child;  
 cin >> child;  
 if (!tree.Exist(child)) {  
 cout << "Error: Parent is not existed" << endl;  
 } else if (node.left\_id == child || node.right\_id == child) {  
 ans = node.ping(child);  
 cout << ans << endl;  
 } else {  
 string message = "ping " + to\_string(child);  
 ans = node.sendstring(message, child);  
 if (ans == "Error: Parent not found") {  
 ans = "Ok: 0";  
 }  
 cout << ans << endl;  
 }  
 } else if (command == "kill") {  
 int child;  
 std::cin >> child;  
 std::string message = "kill";  
 if (!tree.Exist(child)) {  
 std::cout << "Error: Parent is not existed" << std::endl;  
 } else {  
 ans = node.sendstring(message, child);  
 if (ans != "Error: Parent not found") {  
 tree.RemoveFromRoot(child);  
 if (child == node.left\_id) {  
 unbind(node.left, node.left\_port);  
 node.left\_id = -2;  
 ans = "Ok";  
 } else if (child == node.right\_id) {  
 node.right\_id = -2;  
 unbind(node.right, node.right\_port);  
 ans = "Ok";  
 } else {  
 message = "clear " + std::to\_string(child);  
 ans = node.sendstring(message, std::stoi(ans));  
 }  
 std::cout << ans << std::endl;  
 }  
 }  
 } else {  
 std::cout << "Please enter correct command!" << std::endl;  
 }  
  
 }  
  
 node.kill();  
}

**BalancedTree.h**

#ifndef BALANCED\_TREE\_H  
#define BALANCED\_TREE\_H  
  
#include <bits/stdc++.h>  
  
using namespace std;  
  
/\*  
  
 дерево идеально сбалансировано, если для каждого его уровня число узлов  
 в левом и правом поддеревьях отличается не более чем на 1.  
  
\*/  
  
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, 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, 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 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:  
 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;  
 }  
};  
  
#endif

**CalculationNode.h**

#include <bits/stdc++.h>  
#include "ZMQFunctions.h"  
#include "unistd.h"  
  
using namespace std;  
  
class CalculationNode {  
private:  
// управляют сокетами  
 zmq::context\_t context;  
public:  
// для связи между процессами  
  
 zmq::socket\_t left, right, parent;  
 int id, left\_id = -2, right\_id = -2, parent\_id;  
 int left\_port, right\_port, parent\_port;  
  
 CalculationNode(int id, int parent\_port, int parent\_id) :  
 id(id),  
 parent\_port(parent\_port),  
 parent\_id(parent\_id),  
  
// идут парами  
// 1 для связи с вверхом, 2 и 3 для связи с потомками  
  
 left(context, ZMQ\_REQ),  
 right(context, ZMQ\_REQ),  
 parent(context, ZMQ\_REP) {  
 if (id != -1) {  
 connect(parent, parent\_port);  
 }  
 }  
  
 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 {  
 string fail = "Error: can not create the calculation node";  
 return fail;  
 }  
 int fork\_id = fork();  
 if (fork\_id == 0) {  
 if (execl("./server", "server", std::to\_string(child\_id).c\_str(), to\_string(port).c\_str(),  
 to\_string(id).c\_str(), (char \*) NULL) == -1) {  
 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) {  
 string message = "ping " + 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);  
 int result = 0;  
 int amount, number;  
 string\_thread >> amount;  
 for (int i = 0; i < amount; ++i) {  
 string\_thread >> number;  
 result += number;  
 }  
 std::string answer = "Ok: " + std::to\_string(id) + ": " + std::to\_string(result);  
 return answer;  
 }  
  
 std::string treeclear(int child) {  
 if (left\_id == child) {  
 left\_id = -2;  
 unbind(left, left\_port);  
 } else {  
 right\_id = -2;  
 unbind(right, right\_port);  
 }  
 return "Ok";  
 }  
  
 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);  
 }  
  
 ~CalculationNode() {}  
};

**ServerProgram.cpp**

#include <bits/stdc++.h>  
#include "CalculationNode.h"  
#include "ZMQFunctions.h"  
#include "BalancedTree.h"  
  
using namespace std;  
  
int main(int argc, char \*argv[]) {  
 if (argc != 4) {  
 std:: cout << "Usage: 1)./main, 2) child\_id, 3) parent\_port, 4) parent\_id" << std:: endl;  
 exit(EXIT\_FAILURE);  
 }  
 CalculationNode 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 == "send"){  
 int child;  
 std:: string str;  
 request >> child;  
 getline(request, str);  
 str.erase(0, 1);  
 std:: string answer = node.sendstring(str, 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;  
 }  
 else if (command == "clear") {  
 int child;  
 request >> child;  
 std:: string answer = node.treeclear(child);  
 send\_message(node.parent, answer);  
 }  
 }  
 return 0;  
}

**ZMQFunctions.h**

#pragma once  
#include <bits/stdc++.h>  
#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);  
}

**Демонстрация работы программы**

**Enter the command:**

**create id: for creating a new calculation node**

**exec id n n1 n2... n: for calculating a sum**

**ping id: for checking node-availabilty**

**kill id: for killing a calculation node**

**Enter command: create 10**

**Ok: 61**

**Enter command: create 15**

**Ok: 64**

**Enter command: create 11**

**Ok: 67**

**Enter command: create 12**

**Ok: 70**

**Enter command: create 13**

**Ok: 73**

**Enter command: create 14**

**Ok: 76**

**Enter command: exec 14 5 1 2 3 4 5**

**Ok: 14: 15**

**Enter command: ping 15**

**Ok: 1**

**Enter command:**

**Выводы**

Благодаря данной лабораторной работе я узнал, что такое синхронные и асинхронные вычисления. Приобрел практические навыки работы с zero message queue. Закрепил навыки, полученные в предыдущих лабораторных работах.