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»**

Студент: Копылов Михаил Юрьевич

Группа: М8О-201Б-21

Вариант: 1-2-2

Преподаватель: Миронов Евгений Сергеевич

Оценка: \_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2022

**Содержание**

1. Репозиторий
2. Постановка задачи
3. Общие сведения о программе
4. Исходный код
5. Выводы

**Репозиторий**

https://github.com/Mikhail-cWc/OS\_mai/tree/main/lab6

**Постановка задачи**

**Цель работы**

Целью является приобретение практических навыков в:

* Управлении серверами сообщений (№6)
* Применение отложенных вычислений (№7)
* Интеграция программных систем друг с другом (№8)

**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: create, exec, kill, ping.

**Общие сведения о программе**

Программа распределительного узла компилируется из файла control.cpp, программа вычислительного узла компилируется из файла counting.cpp. В программе используется библиотека для работы с потоками и мьютексами, а также сторонняя библиотека для работы с сервером сообщений ZeroMQ.

**Исходный код**

|  |
| --- |
| **control.cpp** |
| #include <unistd.h>  #include <sstream>  #include <set>  #include "zmq\_functions.h"  #include "topology.h"  int main() {  topology network;  std::vector<zmq::socket\_t> branches;  zmq::context\_t context;  std::string cmd;  while (std::cin >> cmd) {  if (cmd == "create") {  int node\_id, parent\_id;  std::cin >> node\_id >> parent\_id;  if (network.find(node\_id) != -1) {  std::cout << "Error: already exists" << std::endl;  }  else if (parent\_id == -1) {  pid\_t pid = fork();  if (pid < 0) {  perror("Can't create new process");  return -1;  }  if (pid == 0) {  if (execl("./counting", "./counting", std::to\_string(node\_id).c\_str(), NULL) < 0) {  perror("Can't execute new process");  return -2;  }  }    branches.emplace\_back(context, ZMQ\_REQ);  branches[branches.size() - 1].setsockopt(ZMQ\_SNDTIMEO, 5000);  bind(branches[branches.size() - 1], node\_id);  send\_message(branches[branches.size() - 1], std::to\_string(node\_id) + "pid");    std::string reply = receive\_message(branches[branches.size() - 1]);  std::cout << reply << std::endl;  network.insert(node\_id, parent\_id);  }  else if (network.find(parent\_id) == -1) {  std::cout << "Error: parent not found" << std::endl;  }  else {  int branch = network.find(parent\_id);  send\_message(branches[branch], std::to\_string(parent\_id) + "create " + std::to\_string(node\_id));  std::string reply = receive\_message(branches[branch]);  std::cout << reply << std::endl;  network.insert(node\_id, parent\_id);  }  }  else if (cmd == "exec") {  int dest\_id;  std::string numbers;  std::cin >> dest\_id;  std::getline(std::cin, numbers);  int branch = network.find(dest\_id);  if (branch == -1) {  std::cout << "ERROR: incorrect node id" << std::endl;  }  else {  send\_message(branches[branch], std::to\_string(dest\_id) + "exec " + numbers);  std::string reply = receive\_message(branches[branch]);  std::cout << reply << std::endl;  }  }  else if (cmd == "kill") {  int id;  std::cin >> id;  int branch = network.find(id);  if (branch == -1) {  std::cout << "ERROR: incorrect node id" << std::endl;  }  else {  bool is\_first = (network.get\_first\_id(branch) == id);  send\_message(branches[branch], std::to\_string(id) + " kill");  std::string reply = receive\_message(branches[branch]);  std::cout << reply << std::endl;  network.erase(id);  if (is\_first) {  unbind(branches[branch], id);  branches.erase(branches.begin() + branch);  }  }  }  else if (cmd == "ping") {  std::set<int> available\_nodes;  for (size\_t i = 0; i < branches.size(); ++i) {  int first\_node\_id = network.get\_first\_id(i);  send\_message(branches[i], std::to\_string(first\_node\_id) + " ping");    std::string received\_message = receive\_message(branches[i]);  std::istringstream reply(received\_message);  int node;  while(reply >> node) {  available\_nodes.insert(node);  }  }  std::cout << "OK: ";  if (available\_nodes.empty()) {  std::cout << "No available nodes" << std::endl;  }  else {  for (auto v : available\_nodes) {  std::cout << v << " ";  }  std::cout << std::endl;  }  }  else if (cmd == "exit") {  for (size\_t i = 0; i < branches.size(); ++i) {  int first\_node\_id = network.get\_first\_id(i);  send\_message(branches[i], std::to\_string(first\_node\_id) + " kill");    std::string reply = receive\_message(branches[i]);  if (reply != "OK") {  std::cout << reply << std::endl;  }  else {  unbind(branches[i], first\_node\_id);  }  }  exit(0);  }  else {  std::cout << "Incorrect cmd" << std::endl;  }  }  } |

|  |
| --- |
| **counting.cpp** |
| #include <unordered\_map>  #include <unistd.h>  #include <sstream>  #include <unordered\_map>    #include "zmq\_functions.h"    int main(int argc, char\* argv[]) {  if (argc != 2 && argc != 3) {  throw std::runtime\_error("Wrong args for counting node");  }  int cur\_id = std::atoi(argv[1]);  int child\_id = -1;  if (argc == 3) {  child\_id = std::atoi(argv[2]);  }  std::unordered\_map<std::string, int> dictionary;  zmq::context\_t context;  zmq::socket\_t parent\_socket(context, ZMQ\_REP);  connect(parent\_socket, cur\_id);  zmq::socket\_t child\_socket(context, ZMQ\_REQ);  child\_socket.setsockopt(ZMQ\_SNDTIMEO, 5000);  if (child\_id != -1) {  bind(child\_socket, child\_id);  }    std::string message;  while (true) {  message = receive\_message(parent\_socket);  std::istringstream request(message);  int dest\_id;  request >> dest\_id;  std::string cmd;  request >> cmd;  if (dest\_id == cur\_id) {  if (cmd == "pid") {  send\_message(parent\_socket, "OK: " + std::to\_string(getpid()));  }  else if (cmd == "create") {  int new\_child\_id;  request >> new\_child\_id;  if (child\_id != -1) {  unbind(child\_socket, child\_id);  }  bind(child\_socket, new\_child\_id);  pid\_t pid = fork();  if (pid < 0) {  perror("Can't create new process");  return -1;  }  if (pid == 0) {  execl("./counting", "./counting", std::to\_string(new\_child\_id).c\_str(), std::to\_string(child\_id).c\_str(), NULL);  perror("Can't execute new process");  return -2;  }  send\_message(child\_socket, std::to\_string(new\_child\_id) + "pid");  child\_id = new\_child\_id;  send\_message(parent\_socket, receive\_message(child\_socket));  }  else if (cmd == "exec") {  int sum = 0;  std::string number;  while (request >> number) {  sum += std::stoi(number);  }  send\_message(parent\_socket, "OK: " + std::to\_string(cur\_id) + ": " + std::to\_string(sum));  }  else if (cmd == "ping") {  std::string reply;  if (child\_id != -1) {  send\_message(child\_socket, std::to\_string(child\_id) + " ping");  std::string msg = receive\_message(child\_socket);  reply += " " + msg;  }  send\_message(parent\_socket, std::to\_string(cur\_id) + reply);  }  else if (cmd == "kill") {  if (child\_id != -1) {  send\_message(child\_socket, std::to\_string(child\_id) + " kill");  std::string msg = receive\_message(child\_socket);  if (msg == "OK") {  send\_message(parent\_socket, "OK");  }  unbind(child\_socket, child\_id);  disconnect(parent\_socket, cur\_id);  break;  }  send\_message(parent\_socket, "OK");  disconnect(parent\_socket, cur\_id);  break;  }  }  else if (child\_id != -1) {  send\_message(child\_socket, message);  send\_message(parent\_socket, receive\_message(child\_socket));  if (child\_id == dest\_id && cmd == "kill") {  child\_id = -1;  }  }  else {  send\_message(parent\_socket, "Error: node is unavailable");  }  }  } |
| **topology.h** |
| #include <list>  #include <stdexcept>  class topology {  private:  std::list<std::list<int>> container;  public:  void insert(int id, int parent\_id) {  if (parent\_id == -1) {  std::list<int> new\_list;  new\_list.push\_back(id);  container.push\_back(new\_list);  }  else {  int list\_id = find(parent\_id);  if (list\_id == -1) {  throw std::runtime\_error("Wrong parent id");  }  auto it1 = container.begin();  std::advance(it1, list\_id);  for (auto it2 = it1->begin(); it2 != it1->end(); ++it2) {  if (\*it2 == parent\_id) {  it1->insert(++it2, id);  return;  }  }  }  }  int find(int id) {  int cur\_list\_id = 0;  for (auto it1 = container.begin(); it1 != container.end(); ++it1) {  for (auto it2 = it1->begin(); it2 != it1->end(); ++it2) {  if (\*it2 == id) {  return cur\_list\_id;  }  }  ++cur\_list\_id;  }  return -1;  }  void erase(int id) {  int list\_id = find(id);  if (list\_id == -1) {  throw std::runtime\_error("Wrong id");  }  auto it1 = container.begin();  std::advance(it1, list\_id);  for (auto it2 = it1->begin(); it2 != it1->end(); ++it2) {  if (\*it2 == id) {  it1->erase(it2, it1->end());  if (it1->empty()) {  container.erase(it1);  }  return;  }  }  }  int get\_first\_id(int list\_id) {  auto it1 = container.begin();  std::advance(it1, list\_id);  if (it1->begin() == it1->end()) {  return -1;  }  return \*(it1->begin());  }  }; |
| **Zmq\_functions.h** |
| #include <zmq.hpp>  #include <iostream>    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;  bool chars\_read;  try {  chars\_read = socket.recv(&message);  }  catch (...) {  chars\_read = false;  }  if (chars\_read == 0) {  return "Error: node is unavailable [zmq\_func]";  }  std::string received\_msg(static\_cast<char\*>(message.data()), message.size());  return received\_msg;  }    void connect(zmq::socket\_t& socket, int id) {  std::string address = "tcp://127.0.0.1:" + std::to\_string(MAIN\_PORT + id);  socket.connect(address);  }  void disconnect(zmq::socket\_t& socket, int id) {  std::string address = "tcp://127.0.0.1:" + std::to\_string(MAIN\_PORT + id);  socket.disconnect(address);  }  void bind(zmq::socket\_t& socket, int id) {  std::string address = "tcp://127.0.0.1:" + std::to\_string(MAIN\_PORT + id);  socket.bind(address);  }  void unbind(zmq::socket\_t& socket, int id) {  std::string address = "tcp://127.0.0.1:" + std::to\_string(MAIN\_PORT + id);  socket.unbind(address);  } |

**Выводы**

Составлена и отлажена программа на языке C++, осуществляющая отложенные вычисления на нескольких вычислительных узлах. Пользователь управляет программой через распределительный узел, который перенаправляет запросы в асинхронном режиме.