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: Кажекин Д.А.

Группа: М8О-207Б-21

Вариант: 31

Преподаватель: Миронов Евгений Сергеевич

Оценка: \_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2022

**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/DKazhekin/OS

**Постановка задачи**

**Цель работы**

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.

**Задание (Вариант 31)**

Тип проверки доступности узлов – pingall

Топология – бинарное дерево общего вида

Тип команд – Локальный таймер (start | stop | time)

**Общие сведения о программе**

Программа состоит из 6 файлов: main.cpp (управляюший узел), client.cpp (получает команды от управляющего узла и выполняет их), timer.cpp / timer.h (реализация таймера), tree.cpp / tree.h (реализация дерева)

1. create id() – вставка вычислительного узла в бинарное дерево
2. exec id subcmd – отправка команды вычислительному узлу
3. kill id – удаление вычислительного узла и всех его дочерних узлов из дерева
4. pingall – все вычислительные узлы подтверждают свою работоспособность

**Общий метод и алгоритм решения**

С помощью библиотеки ZMQ реализованы функции отправки и получения сообщения по сокетам. Реализовано бинарное дерево поиска (топология), по которой распределяются все вычислительные узлы.

**Исходный код**

**Main.cpp**

**#include <iostream>**

**#include <unistd.h>**

**#include <string>**

**#include <vector>**

**#include <set>**

**#include <sstream>**

**#include <csignal>**

**#include "zmq.hpp"**

**#include "tree.h"**

**const int WAIT\_TIME = 1000;**

**const int PORT\_BASE = 5050;**

**bool send\_message(zmq::socket\_t &socket, const std::string &message\_string)**

**{**

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

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

**return socket.send(message);**

**}**

**std::string recieve\_message(zmq::socket\_t &socket)**

**{**

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

**bool ok = false;**

**try**

**{**

**ok = socket.recv(&message);**

**}**

**catch (...)**

**{**

**ok = false;**

**}**

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

**if (recieved\_message.empty() || !ok)**

**{**

**return "Error: Node is not available";**

**}**

**return recieved\_message;**

**}**

**void create\_node(int id, int port)**

**{**

**char\* arg0 = strdup("./client");**

**char\* arg1 = strdup((std::to\_string(id)).c\_str());**

**char\* arg2 = strdup((std::to\_string(port)).c\_str());**

**char\* args[] = {arg0, arg1, arg2, NULL};**

**execv("./client", args);**

**}**

**std::string get\_port\_name(const int port)**

**{**

**return "tcp://127.0.0.1:" + std::to\_string(port);**

**}**

**bool is\_number(std::string val)**

**{**

**try**

**{**

**int tmp = std::stoi(val);**

**return true;**

**}**

**catch(std::exception &e)**

**{**

**std::cout << "Error: " << e.what() << "\n";**

**return false;**

**}**

**}**

**int main()**

**{**

**Tree T;**

**std::string command;**

**int child\_pid = 0;**

**int child\_id = 0;**

**zmq::context\_t context(1);**

**zmq::socket\_t main\_socket(context, ZMQ\_REQ);**

**std::cout << "Commands:\n";**

**std::cout << "create id\n";**

**std::cout << "exec id subcommand (start/stop/time)\n";**

**std::cout << "kill id\n";**

**std::cout << "pingall\n";**

**std::cout << "exit\n" << std::endl;**

**while(1)**

**{**

**std::cin >> command;**

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

**{**

**size\_t node\_id = 0;**

**std::string str = " ";**

**std::string result = " ";**

**std::cin >> str;**

**if (!is\_number(str))**

**{**

**continue;**

**}**

**node\_id = stoi(str);**

**if (child\_pid == 0)**

**{**

**main\_socket.bind(get\_port\_name(PORT\_BASE + node\_id));**

**child\_pid = fork();**

**if (child\_pid == -1)**

**{**

**std::cout << "Unable to create first worker node\n";**

**child\_pid = 0;**

**exit(1);**

**}**

**else if (child\_pid == 0)**

**{**

**create\_node(node\_id, PORT\_BASE + node\_id);**

**}**

**else**

**{**

**child\_id = node\_id;**

**send\_message(main\_socket,"pid");**

**result = recieve\_message(main\_socket);**

**}**

**}**

**else**

**{**

**std::string msg\_s = "create " + std::to\_string(node\_id);**

**send\_message(main\_socket, msg\_s);**

**result = recieve\_message(main\_socket);**

**}**

**if (result.substr(0, 2) == "Ok")**

**{**

**T.push(node\_id);**

**}**

**std::cout << result << "\n";**

**}**

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

**{**

**int node\_id = 0;**

**std::string str = " ";**

**std::cin >> str;**

**if (!is\_number(str))**

**{**

**continue;**

**}**

**node\_id = stoi(str);**

**if (child\_pid == 0)**

**{**

**std::cout << "Error: Not found\n";**

**continue;**

**}**

**if (node\_id == child\_id)**

**{**

**kill(child\_pid, SIGTERM);**

**kill(child\_pid, SIGKILL);**

**child\_id = 0;**

**child\_pid = 0;**

**T.kill(node\_id);**

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

**continue;**

**}**

**std::string message\_string = "kill " + std::to\_string(node\_id);**

**send\_message(main\_socket, message\_string);**

**std::string recieved\_message = recieve\_message(main\_socket);**

**if (recieved\_message.substr(0, std::min<int>(recieved\_message.size(), 2)) == "Ok")**

**{**

**T.kill(node\_id);**

**}**

**std::cout << recieved\_message << "\n";**

**}**

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

**{**

**std::string id\_str = " ";**

**std::string subcommand = " ";**

**int id = 0;**

**std::cin >> id\_str >> subcommand;**

**if (!is\_number(id\_str))**

**{**

**continue;**

**}**

**id = stoi(id\_str);**

**if ((subcommand != "start") && (subcommand != "stop") && (subcommand != "time"))**

**{**

**std::cout << "Wrong subcommandmand\n";**

**continue;**

**}**

**std::string message\_string = "exec " + std::to\_string(id) + " " + subcommand;**

**send\_message(main\_socket, message\_string);**

**std::string recieved\_message = recieve\_message(main\_socket);**

**std::cout << recieved\_message << "\n";**

**}**

**else if (command == "pingall")**

**{**

**if (child\_id == 0)**

**{**

**std::cout << "No childs to ping!\n";**

**}**

**else**

**{**

**std::string message\_string = command;**

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

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

**main\_socket.send(message);**

**main\_socket.recv(&message);**

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

**if (recieved\_message == "OK")**

**{**

**std::cout << "OK: -1\n";**

**}**

**else**

**{**

**std::cout << "OK:" << recieved\_message;**

**}**

**}**

**}**

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

**{**

**int n = system("killall client");**

**break;**

**}**

**}**

**return 0;**

**}**

**Client.cpp**

**#include <iostream>**

**#include <unistd.h>**

**#include <string>**

**#include <sstream>**

**#include <exception>**

**#include <signal.h>**

**#include "zmq.hpp"**

**#include "timer.h"**

**const int WAIT\_TIME = 1000;**

**const int PORT\_BASE = 5050;**

**bool send\_message(zmq::socket\_t &socket, const std::string &message\_string)**

**{**

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

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

**return socket.send(message);**

**}**

**std::string recieve\_message(zmq::socket\_t &socket)**

**{**

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

**bool ok = false;**

**try**

**{**

**ok = socket.recv(&message);**

**}**

**catch (...)**

**{**

**ok = false;**

**}**

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

**if (recieved\_message.empty() || !ok)**

**{**

**return "Error: Node is not available";**

**}**

**return recieved\_message;**

**}**

**void create\_node(int id, int port)**

**{**

**char\* arg0 = strdup("./client");**

**char\* arg1 = strdup((std::to\_string(id)).c\_str());**

**char\* arg2 = strdup((std::to\_string(port)).c\_str());**

**char\* args[] = {arg0, arg1, arg2, nullptr};**

**execv("./client", args);**

**}**

**std::string get\_port\_name(const int port)**

**{**

**return "tcp://127.0.0.1:" + std::to\_string(port);**

**}**

**void rl\_create(zmq::socket\_t& parent\_socket, zmq::socket\_t& socket, int& create\_id, int& id, int& pid)**

**{**

**if (pid == -1)**

**{**

**send\_message(parent\_socket, "Error: Cannot fork");**

**pid = 0;**

**}**

**else if (pid == 0)**

**{**

**create\_node(create\_id,PORT\_BASE + create\_id);**

**}**

**else**

**{**

**id = create\_id;**

**send\_message(socket, "pid");**

**send\_message(parent\_socket, recieve\_message(socket));**

**}**

**}**

**void rl\_kill(zmq::socket\_t& parent\_socket, zmq::socket\_t& socket, int& delete\_id, int& id, int& pid, std::string& request\_string)**

**{**

**if (id == 0)**

**{**

**send\_message(parent\_socket, "Error: Not found");**

**}**

**else if (id == delete\_id)**

**{**

**send\_message(socket, "kill\_children");**

**recieve\_message(socket);**

**kill(pid,SIGTERM);**

**kill(pid,SIGKILL);**

**id = 0;**

**pid = 0;**

**send\_message(parent\_socket, "Ok");**

**}**

**else**

**{**

**send\_message(socket, request\_string);**

**send\_message(parent\_socket, recieve\_message(socket));**

**}**

**}**

**void rl\_exec(zmq::socket\_t& parent\_socket, zmq::socket\_t& socket, int& id, int& pid, std::string& request\_string)**

**{**

**if (pid == 0)**

**{**

**std::string recieve\_message = "Error:" + std::to\_string(id);**

**recieve\_message += ": Not found";**

**send\_message(parent\_socket, recieve\_message);**

**}**

**else**

**{**

**send\_message(socket, request\_string);**

**send\_message(parent\_socket, recieve\_message(socket));**

**}**

**}**

**void exec(std::istringstream& command\_stream, zmq::socket\_t& parent\_socket, zmq::socket\_t& left\_socket,**

**zmq::socket\_t& right\_socket, int& left\_pid, int& right\_pid, int& id, std::string& request\_string, Timer\* timer)**

**{**

**std::string subcommand;**

**int exec\_id;**

**command\_stream >> exec\_id;**

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

**{**

**command\_stream >> subcommand;**

**std::string recieve\_message = "";**

**if (subcommand == "start")**

**{**

**timer->start\_timer();**

**recieve\_message = "Ok:" + std::to\_string(id);**

**send\_message(parent\_socket, recieve\_message);**

**}**

**else if (subcommand == "stop")**

**{**

**timer->stop\_timer();**

**recieve\_message = "Ok:" + std::to\_string(id);**

**send\_message(parent\_socket, recieve\_message);**

**}**

**else if (subcommand == "time")**

**{**

**recieve\_message = "Ok:" + std::to\_string(id) + ": ";**

**recieve\_message += std::to\_string(timer->get\_time());**

**send\_message(parent\_socket, recieve\_message);**

**}**

**}**

**else if (exec\_id < id)**

**{**

**rl\_exec(parent\_socket, left\_socket, exec\_id,**

**left\_pid, request\_string);**

**}**

**else**

**{**

**rl\_exec(parent\_socket, right\_socket, exec\_id,**

**right\_pid, request\_string);**

**}**

**}**

**void pingall(zmq::socket\_t& parent\_socket, int& id, zmq::socket\_t& left\_socket, zmq::socket\_t& right\_socket,int& left\_pid, int& right\_pid)**

**{**

**std::ostringstream res;**

**std::string left\_res;**

**std::string right\_res;**

**res << std::to\_string(id);**

**if (left\_pid != 0)**

**{**

**send\_message(left\_socket, "pingall");**

**left\_res = recieve\_message(left\_socket);**

**}**

**if (right\_pid != 0)**

**{**

**send\_message(right\_socket, "pingall");**

**right\_res = recieve\_message(right\_socket);**

**}**

**if (!left\_res.empty() && left\_res.substr(0, std::min<int>(left\_res.size(),5) ) != "Error")**

**{**

**res << " " << left\_res;**

**}**

**if ((!right\_res.empty()) && (right\_res.substr(0, std::min<int>(right\_res.size(),5) ) != "Error"))**

**{**

**res << " "<< right\_res;**

**}**

**send\_message(parent\_socket, res.str());**

**}**

**void kill\_children(zmq::socket\_t& parent\_socket, zmq::socket\_t& left\_socket, zmq::socket\_t& right\_socket, int& left\_pid, int& right\_pid)**

**{**

**if (left\_pid == 0 && right\_pid == 0)**

**{**

**send\_message(parent\_socket, "Ok");**

**}**

**else**

**{**

**if (left\_pid != 0)**

**{**

**send\_message(left\_socket, "kill\_children");**

**recieve\_message(left\_socket);**

**kill(left\_pid,SIGTERM);**

**kill(left\_pid,SIGKILL);**

**}**

**if (right\_pid != 0)**

**{**

**send\_message(right\_socket, "kill\_children");**

**recieve\_message(right\_socket);**

**kill(right\_pid,SIGTERM);**

**kill(right\_pid,SIGKILL);**

**}**

**send\_message(parent\_socket, "Ok");**

**}**

**}**

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

**{**

**Timer timer;**

**int id = std::stoi(argv[1]);**

**int parent\_port = std::stoi(argv[2]);**

**zmq::context\_t context(3);**

**zmq::socket\_t parent\_socket(context, ZMQ\_REP);**

**parent\_socket.connect(get\_port\_name(parent\_port));**

**int left\_pid = 0;**

**int right\_pid = 0;**

**int left\_id = 0;**

**int right\_id = 0;**

**zmq::socket\_t left\_socket(context, ZMQ\_REQ);**

**zmq::socket\_t right\_socket(context, ZMQ\_REQ);**

**while(1)**

**{**

**std::string request\_string = recieve\_message(parent\_socket);**

**std::istringstream command\_stream(request\_string);**

**std::string command;**

**command\_stream >> command;**

**if (command == "id")**

**{**

**std::string parent\_string = "Ok:" + std::to\_string(id);**

**send\_message(parent\_socket, parent\_string);**

**}**

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

**{**

**std::string parent\_string = "Ok:" + std::to\_string(getpid());**

**send\_message(parent\_socket, parent\_string);**

**}**

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

**{**

**int create\_id;**

**command\_stream >> create\_id;**

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

**{**

**std::string message\_string = "Error: Already exists";**

**send\_message(parent\_socket, message\_string);**

**}**

**else if (create\_id < id)**

**{**

**if (left\_pid == 0)**

**{**

**left\_socket.bind(get\_port\_name(PORT\_BASE + create\_id));**

**left\_pid = fork();**

**rl\_create(parent\_socket, left\_socket, create\_id, left\_id, left\_pid);**

**}**

**else**

**{**

**send\_message(left\_socket, request\_string);**

**send\_message(parent\_socket, recieve\_message(left\_socket));**

**}**

**}**

**else**

**{**

**if (right\_pid == 0)**

**{**

**right\_socket.bind(get\_port\_name(PORT\_BASE + create\_id));**

**right\_pid = fork();**

**rl\_create(parent\_socket, right\_socket, create\_id, right\_id, right\_pid);**

**}**

**else**

**{**

**send\_message(right\_socket, request\_string);**

**send\_message(parent\_socket, recieve\_message(right\_socket));**

**}**

**}**

**}**

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

**{**

**int delete\_id;**

**command\_stream >> delete\_id;**

**if (delete\_id < id)**

**{**

**rl\_kill(parent\_socket, left\_socket, delete\_id, left\_id, left\_pid, request\_string);**

**}**

**else**

**{**

**rl\_kill(parent\_socket, right\_socket, delete\_id, right\_id, right\_pid, request\_string);**

**}**

**}**

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

**{**

**exec(command\_stream, parent\_socket, left\_socket, right\_socket, left\_pid, right\_pid, id, request\_string, &timer);**

**}**

**else if (command == "pingall")**

**{**

**pingall(parent\_socket, id, left\_socket, right\_socket, left\_pid, right\_pid);**

**}**

**else if (command == "kill\_children")**

**{**

**kill\_children(parent\_socket, left\_socket, right\_socket, left\_pid, right\_pid);**

**}**

**if (parent\_port == 0)**

**{**

**break;**

**}**

**}**

**return 0;**

**}**

**Демонстрация работы программы**

deniskazhekin@Deniss-Air lab6 % make all

g++ -Wno-unused-variable -c main.cpp -o main.o

g++ -Wno-unused-variable main.o tree.o -o main -lzmq -L/opt/homebrew/Cellar/zeromq/4.3.4/lib -std=c++11

g++ -Wno-unused-variable -c client.cpp -o client.o

g++ -Wno-unused-variable client.o timer.o -o client -lzmq -L/opt/homebrew/Cellar/zeromq/4.3.4/lib -std=c++11

deniskazhekin@Deniss-Air lab6 % ./main

Commands:

create id

exec id subcommand (start/stop/time)

kill id

pingall

exit

create 2

Ok:12939

create 5

Ok:12945

create 7

Ok:12956

pingall

OK:2 5 7

exec 5 start

Ok:5

exec 5 time

Ok:5: 5630

exec 5 stop

Ok:5

exec 5 time

Ok:5: 12600

kill 5

Ok

pingall

OK:2

create 7

Ok:13186

create 3

Ok:13201

kill 7

Ok

pingall

OK:2kill 2

Ok

pingall

No childs to ping!

**Выводы**

Выполняя лабораторную работу, я освоил основы библиотеки ZMQ, а также познакомился с очередями сообщений.