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: Ивченко Анна Владимировна

Группа: М8О-208Б-20

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/Anetta123/OS/tree/main/laba6-8/src>

**Постановка задачи**

Реализовать распределенную систему по обработке запросов. В данной 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 сервера сообщений zmq. Также в данной системе необходимо предусмотреть проверку доступности узлов в соответствии с вариантом.

**Вариант задания:**

36. Топология — бинарное дерево. Тип вычислительной команды — сумма n чисел. Тип проверки узлов на доступность — пинг всех узлов.

**Общие сведения о программе:**

Программа состоит из двух файлов, которые компилируются в исполнительные файлы(которые представляют управляющий и вычислительные узлы). Общение между процессами происходит с помощью библиотеки zmq.

**Общий метод и алгоритм решения:**

* Управляющий узел принимает команды, обрабатывает их и пересылает дочерним узлам(или выводит сообщение об ошибке).
* Дочерние узлы проверяют, может ли быть команда выполнена в данном узле, если нет, то команда пересылается в один из дочерних узлов, из которого возвращается некоторое сообщение(об успехе или об ошибке), которое потом пересылается обратно по дереву.
* Для корректной проверки на доступность узлов, используется дерево, эмулирующее поведение узлов в данной топологии(например, при удалении узла, удаляются все его потомки).
* Если узел недоступен, то по истечении таймаута будет сгенерировано сообщение о недоступности узла и оно будет передано вверх по дереву, к управляющему узлу.

При удалении узла, все его потомки рекурсивно уничтожаются.

**Исходный код:**

**Main\_prog.cpp**

#include "zmq.hpp"

#include <sstream>

#include <string>

#include <iostream>

#include <zconf.h>

#include <vector>

#include <signal.h>

#include <sstream>

#include <set>

#include <algorithm>

// g++ main\_prog.cpp -lzmq -o main\_prog -w

using namespace std;

int main(){

zmq::context\_t context(1);

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

string adr = "tcp://127.0.0.1:500";

string command;

int child\_id = 0;

while(1){

cout << "Please, enter command\n";

cin >> command;

if(command == "create"){

if(child\_id == 0){

int id;

cin >> id;

int id\_tmp = id - 1;

while(1){

try {

main\_socket.bind(adr + to\_string(++id\_tmp));

break;

}

catch(...) {}

}

string new\_adr = adr + to\_string(id\_tmp);

char\* adr\_ = new char[new\_adr.size() + 1];

memcpy(adr\_, new\_adr.c\_str(), new\_adr.size() + 1);

char\* id\_ = new char[to\_string(id).size() + 1];

memcpy(id\_, to\_string(id).c\_str(), to\_string(id).size() + 1);

char\* args[] = {"./child\_node", adr\_, id\_, NULL};

int id2 = fork();

if (id2 == -1) {

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

id = 0;

exit(1);

}

else if(id2 == 0){

execv("./child\_node", args);

}

else {

child\_id = id;

}

zmq::message\_t message;

main\_socket.recv(&message);

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

cout << recieved\_message << "\n";

delete [] adr\_;

delete [] id\_;

}

else {

int id;

cin >> id;

string message\_string = command + " " + to\_string(id);

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

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

main\_socket.send(message);

// catch message from new node

main\_socket.recv(&message);

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

cout << recieved\_message << "\n";

}

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

int id, value;

string name;

cin >> id >> name;

string s;

char q;

while((q = getchar()) != '\n'){

s += q;

}

if(s == ""){

string message\_string = command + " " + to\_string(id) + " " + name;

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

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

main\_socket.send(message);

// return value from map

main\_socket.recv(&message);

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

cout << recieved\_message << "\n";

} else {

value = stoi(s);

string message\_string = command + " " + to\_string(id) + " " + name + " " + to\_string(value);

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

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

main\_socket.send(message);

// add new element to map

main\_socket.recv(&message);

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

cout << recieved\_message << "\n";

}

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

int id;

cin >> id;

string message\_string = command + " " + to\_string(id);

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

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

main\_socket.send(message);

// receive answer from child

main\_socket.recv(&message);

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

cout << recieved\_message << "\n";

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

int id;

cin >> id;

if(child\_id == 0){

cout << "Error: there isn't nodes\n";

} else if(child\_id == id){

string kill\_message = command + " " + to\_string(id);

zmq::message\_t message(kill\_message.size());

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

main\_socket.send(message);

main\_socket.recv(message);

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

cout << received\_message << "\n";

cout << "Tree deleted successfully\n";

return 0;

} else {

string kill\_message = command + " " + to\_string(id);

zmq::message\_t message(kill\_message.size());

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

main\_socket.send(message);

main\_socket.recv(&message);

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

cout << received\_message << "\n";

}

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

if(child\_id){

string kill\_message = "DIE";

zmq::message\_t message(kill\_message.size());

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

main\_socket.send(message);

cout << "Tree was deleted\n";

}

main\_socket.close();

context.close();

break;

} else {

cout << "Error: incorrect command\n";

}

}

}

**Child\_node2.cpp**

#include "zmq.hpp"

#include <sstream>

#include <string>

#include <iostream>

#include <zconf.h>

#include <vector>

#include <signal.h>

#include <fstream>

#include <algorithm>

#include <map>

// g++ child\_node\_2.cpp -lzmq -o child\_node -w

using namespace std;

void send\_message(string message\_string, zmq::socket\_t& socket){

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

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

if(!socket.send(message\_back))

{

cout << "Error: can't send message from node with pid " << getpid() << "\n";

}

}

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

{

string adr = argv[1];

zmq::context\_t context(1);

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

main\_socket.connect(argv[1]);

send\_message("OK: " + to\_string(getpid()), main\_socket);

int id = stoi(argv[2]); // id of this node

map<string, int> m;

int left\_id = 0;

int right\_id = 0;

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

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

zmq::socket\_t left\_socket(context\_l, ZMQ\_REP);

string adr\_left = "tcp://127.0.0.1:500";

zmq::socket\_t right\_socket(context\_r, ZMQ\_REP);

string adr\_right = "tcp://127.0.0.1:500";

while(1)

{

zmq::message\_t message\_main;

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

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

string command;

for(int i = 0; i < recieved\_message.size(); ++i){

if(recieved\_message[i] != ' '){

command += recieved\_message[i];

} else {

break;

}

}

if(command == "exec"){

int id\_proc; // id of node for adding

string id\_proc\_, value\_;

string key;

int value;

for(int i = 5; i < recieved\_message.size(); ++i){

if(recieved\_message[i] != ' '){

id\_proc\_ += recieved\_message[i];

} else {

break;

}

}

id\_proc = stoi(id\_proc\_);

if(id\_proc == id){ // id == proc\_id

for(int i = 6 + id\_proc\_.size(); i < recieved\_message.size(); ++i){

if(recieved\_message[i] != ' '){

key += recieved\_message[i];

} else {

break;

}

}

for(int i = 7 + id\_proc\_.size() + key.size(); i < recieved\_message.size(); ++i){

if(recieved\_message[i] != ' ' || recieved\_message[i] != '\n'){

value\_ += recieved\_message[i];

} else {

break;

}

}

if(value\_ == ""){

if(m.count(key)){

int value\_map = m[key];

send\_message("OK:" + id\_proc\_ + ":" + to\_string(m[key]), main\_socket);

} else {

cout << key;

send\_message("OK:" + id\_proc\_ + ": \'" + key + "\' not found", main\_socket);

}

} else {

m[key] = stoi(value\_);

send\_message("OK:" + id\_proc\_, main\_socket);

}

} else {

if(id > id\_proc){

if(left\_id == 0){ // if node not exists

string message\_string = "Error:id: Not found";

send\_message("Error:id: Not found", main\_socket);

} else {

zmq::message\_t message(recieved\_message.size());

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

if(!left\_socket.send(message)){

cout << "Error: can't send message to left node from node with pid: " << getpid() << "\n";

}

// catch and send to parent

if(!left\_socket.recv(&message)){

cout << "Error: can't receive message from left node in node with pid: " << getpid() << "\n";

}

if(!main\_socket.send(message)){

cout << "Error: can't send message to main node from node with pid: " << getpid() << "\n";

}

}

} else {

if(right\_id == 0){ // if node not exists

string message\_string = "Error:id: Not found";

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

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

if(!main\_socket.send(message)){

cout << "Error: can't send message to main node from node with pid: " << getpid() << "\n";

}

} else {

zmq::message\_t message(recieved\_message.size());

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

if(!right\_socket.send(message)){

cout << "Error: can't send message to right node from node with pid: " << getpid() << "\n";

}

// catch and send to parent

if(!right\_socket.recv(&message)){

cout << "Error: can't receive message from left node in node with pid: " << getpid() << "\n";

}

if(!main\_socket.send(message)){

cout << "Error: can't send message to main node from node with pid: " << getpid() << "\n";

}

}

}

}

} else if(command == "create"){

int id\_proc; // id of node for creating

string id\_proc\_;

for(int i = 7; i < recieved\_message.size(); ++i){

if(recieved\_message[i] != ' '){

id\_proc\_ += recieved\_message[i];

} else {

break;

}

}

id\_proc = stoi(id\_proc\_);

if(id\_proc == id){

send\_message("Error: Already exists", main\_socket);

} else if(id\_proc > id){

if(right\_id == 0){ // there is not right node

right\_id = id\_proc;

int right\_id\_tmp = right\_id - 1;

while(1){

try {

right\_socket.bind(adr\_right + to\_string(++right\_id\_tmp));

break;

} catch(...) {

}

}

adr\_right += to\_string(right\_id\_tmp);

char\* adr\_right\_ = new char[adr\_right.size() + 1];

memcpy(adr\_right\_, adr\_right.c\_str(), adr\_right.size() + 1);

char\* right\_id\_ = new char[to\_string(right\_id).size() + 1];

memcpy(right\_id\_, to\_string(right\_id).c\_str(), to\_string(right\_id).size() + 1);

char\* args[] = {"./child\_node", adr\_right\_, right\_id\_, NULL};

int f = fork();

if(f == 0){

execv("./child\_node", args);

} else if (f == -1){

cout << "Error in forking in node with pid: " << getpid() << "\n";

} else {

// catch message from new node

zmq::message\_t message\_from\_node;

if(!right\_socket.recv(&message\_from\_node)){

cout << "Error: can't receive message from right node in node with pid:" << getpid() << "\n";

}

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

if(!main\_socket.send(message\_from\_node)){

cout << "Error: can't send message to main node from node with pid:" << getpid() << "\n";

}

}

delete [] adr\_right\_;

delete [] right\_id\_;

} else { // send task to right node

send\_message(recieved\_message, right\_socket);

// catch and send to parent

zmq::message\_t message;

if(!right\_socket.recv(&message)){

cout << "Error: can't receive message from left node in node with pid: " << getpid() << "\n";

}

if(!main\_socket.send(message)){

cout << "Error: can't send message to main node from node with pid: " << getpid() << "\n";

}

}

} else {

if(left\_id == 0){ // there is not left node

left\_id = id\_proc;

int left\_id\_tmp = left\_id - 1;

while(1){

try {

left\_socket.bind(adr\_left + to\_string(++left\_id\_tmp));

break;

} catch(...) {

}

}

adr\_left += to\_string(left\_id\_tmp);

char\* adr\_left\_ = new char[adr\_left.size() + 1];

memcpy(adr\_left\_, adr\_left.c\_str(), adr\_left.size() + 1);

char\* left\_id\_ = new char[to\_string(left\_id).size() + 1];

memcpy(left\_id\_, to\_string(left\_id).c\_str(), to\_string(left\_id).size() + 1);

char\* args[] = {"./child\_node", adr\_left\_, left\_id\_, NULL};

int f = fork();

if(f == 0){

execv("./child\_node", args);

} else if(f == -1){

cout << "Error in forking in node with pid: " << getpid() << "\n";

} else {

// catch message from new node

zmq::message\_t message\_from\_node;

if(!left\_socket.recv(&message\_from\_node)){

cout << "Error: can't receive message from left node in node with pid:" << getpid() << "\n";

}

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

// send message to main node

if(!main\_socket.send(message\_from\_node)){

cout << "Error: can't send message to main node from node with pid:" << getpid() << "\n";

}

}

delete [] adr\_left\_;

delete [] left\_id\_;

} else { // send task to left node

send\_message(recieved\_message, left\_socket);

// catch and send to parent

zmq::message\_t message;

if(!left\_socket.recv(&message)){

cout << "Error: can't receive message from left node in node with pid: " << getpid() << "\n";

}

if(!main\_socket.send(message)){

cout << "Error: can't send message to main node from node with pid: " << getpid() << "\n";

}

}

}

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

int id\_proc; // id of node for creating

string id\_proc\_;

for(int i = 5; i < recieved\_message.size(); ++i){

if(recieved\_message[i] != ' '){

id\_proc\_ += recieved\_message[i];

} else {

break;

}

}

id\_proc = stoi(id\_proc\_);

if(id\_proc == id){

send\_message("OK: 1", main\_socket);

} else if(id\_proc < id) {

if(left\_id == 0){

send\_message("OK: 0", main\_socket);

} else {

left\_socket.send(message\_main);

zmq::message\_t answ;

left\_socket.recv(&answ);

main\_socket.send(answ);

}

} else if(id\_proc > id) {

if(right\_id == 0){

send\_message("OK: 0", main\_socket);

} else {

right\_socket.send(message\_main);

zmq::message\_t answ;

right\_socket.recv(&answ);

main\_socket.send(answ);

}

}

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

int id\_proc; // id of node for killing

string id\_proc\_;

for(int i = 5; i < recieved\_message.size(); ++i){

if(recieved\_message[i] != ' '){

id\_proc\_ += recieved\_message[i];

} else {

break;

}

}

id\_proc = stoi(id\_proc\_);

if(id\_proc > id){

if(right\_id == 0){

send\_message("Error: there isn`t node with this id", main\_socket);

} else {

if(right\_id == id\_proc){

send\_message("Ok: " + to\_string(right\_id), main\_socket);

send\_message("DIE", right\_socket);

right\_socket.unbind(adr\_right);

adr\_right = "tcp://127.0.0.1:500";

right\_id = 0;

} else {

right\_socket.send(message\_main);

zmq::message\_t message;

right\_socket.recv(&message);

main\_socket.send(message);

}

}

} else if(id\_proc < id){

if(left\_id == 0){

send\_message("Error: there isn`t node with this id", main\_socket);

} else {

if(left\_id == id\_proc){

send\_message("Ok: " + to\_string(left\_id), main\_socket);

send\_message("DIE", left\_socket);

left\_socket.unbind(adr\_left);

adr\_left = "tcp://127.0.0.1:500";

left\_id = 0;

} else {

left\_socket.send(message\_main);

zmq::message\_t message;

left\_socket.recv(&message);

main\_socket.send(message);

}

}

}

} else if (command == "DIE") {

if (left\_id){

send\_message("DIE", left\_socket);

left\_socket.unbind(adr\_left);

adr\_left = "tcp://127.0.0.1:500";

left\_id = 0;

}

if (right\_id){

send\_message("DIE", right\_socket);

right\_socket.unbind(adr\_right);

adr\_right = "tcp://127.0.0.1:500";

right\_id = 0;

}

main\_socket.unbind(adr);

return 0;

}

}

}

**Демонстрация работы программы**

Please, enter command

create 9

OK: 3408

Please, enter command

create 7

OK: 3415

Please, enter command

create 1

OK: 3422

Please, enter command

create 12

OK: 3429

Please, enter command

create 10

OK: 3436

Please, enter command

create 20

OK: 3443

Please, enter command

exec 10 MyVar

OK:10: 'MyVar' not found

Please, enter command

exec 10 MyVar 5

OK:10

Please, enter command

exec 12 MyVar

OK:12: 'MyVar' not found

Please, enter command

exec 10 MyVar

OK:10:5

Please, enter command

exec 10 MyVar

OK:10:5

Please, enter command

ping 10

OK: 1

Please, enter command

ping 17

OK: 0

Please, enter command

exit

Tree was deleted

**Выводы**

Данная лабораторная работа оказалось очень сложной, но очень интересной. В ней сразу применяем знания, полученные в ходе выполнения предыдущих лабораторных работ, так как здесь и многопоточность, и межпроцессорное взаимодействие, и синхронизация потоков. А помимо всего этого также разобрались с дополнительной библиотекой (zmq).