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: Литовченко А.A.

Группа: М8О-207Б-21

Вариант: 32

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/Annalitov/OS/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, remove, ping.

Вариант 32

Топология: бинарное дерево, набор команд: локальный таймер, проверка: ping

**Общие сведения о программе**

Программа распределительного узла компилируется из файла distribution.cpp(клиент), программа вычислительного узла компилируется из файла computing.cpp(сервер). В программе используется библиотека для работы с потоками и мьютексами, а также сторонняя библиотека для работы с сервером сообщений ZeroMQ.

**Общий метод и алгоритм решения**

Распределительный узел хранит в себе двоичное дерево, элементами которого являются async\_node – структуры, хранящие в себе id вычислительного узла, порт для связи с ним, поток отправляющий/принимающий запросы/ответы к узлу, очередь отправленных на узел запросов и мьютекс, обеспечивающий возможность работать с этой очередью в несколько потоков.

В функции main запускается бесконечный цикл обработки пользовательских запросов, при получении запроса он отправляется в очередь обработчика соответствующего узла, откуда позже будет извлечён потоком обработки и переслан требуемому узлу (обеспечение асинхронности). При удалении узла у записи, соответствующей удаляемому узлу устанавливается переменная активности в false, благодаря чему обработчик, отправив на узел запрос удаления и получив ответ, выходит из цикла и очищает память от уже ненужной записи. Подобные действия применяются и ко всем дочерним узлам удаляемого.

Ping выполняет проверку доступности узла: если монитор сокета получает сигнал ZMQ\_EVENT\_CONNECTED, то узел считается доступным, если же вместо этого приходит ZMQ\_EVENT\_CONNECT\_RETRIED, то узел считается недоступным.

**Исходный код**

**node.h**

#ifndef NODE\_H

#define NODE\_H

#include "zmq.h"

#include "string.h"

#include "unistd.h"

#include "stdlib.h"

#include "pthread.h"

#include <iostream>

#include <queue>

#include <vector>

std::string protocol = "tcp://localhost:";

const int MIN\_PORT = 1024;

void\* async\_node\_thd(void\*);

const int CREATE = 1;

const int EXEC = 0;

const int REMOVE = -1;

struct async\_node

{

int id, act;

std::string port;

bool active;

async\_node\* L;

async\_node\* R;

pthread\_mutex\_t mutex;

pthread\_t thd;

std::queue <std::vector <int>> q;

async\_node(int i)

{

id = i;

port = protocol + std::to\_string(i);

active = true;

L = nullptr;

R = nullptr;

act = pthread\_mutex\_init(&mutex, NULL);

if (act != 0){

std::cout << "Error:" << id - MIN\_PORT << ": Gateway mutex error\n";

return;

}

act = pthread\_create(&thd, NULL, async\_node\_thd, this);

if (act != 0){

std::cout << "Error:" << id - MIN\_PORT << ": Gateway thread error\n";

return;

}

act = pthread\_detach(thd);

if(act != 0){

std::cout << "Error:" << id << ": Gateway thread error\n";

return;

}

}

void make\_query(std::vector <int> v)

{

act = pthread\_mutex\_lock(&mutex);

if (act != 0){

std::cout << "Error:" << id - MIN\_PORT << ": Gateway mutex lock error\n";

active = false;

return;

}

q.push(v);

act = pthread\_mutex\_unlock(&mutex);

if (act != 0){

std::cout << "Error:" << id - MIN\_PORT << ": Gateway mutex unlock error\n";

active = false;

}

}

~async\_node()

{

pthread\_mutex\_destroy(&mutex);

}

};

void\* async\_node\_thd(void\* ptr)

{

int act;

async\_node\* node = (async\_node\*)ptr;

void\* context = zmq\_ctx\_new();

void \*req = zmq\_socket(context, ZMQ\_REQ);

act = zmq\_connect(req, node->port.c\_str());

if (act == -1){

std::cout << "Error: Connection with" << node->id - MIN\_PORT << "\n";

}

while (node->active || !node->q.empty())

{

if (node->q.empty()) {

continue;

}

act = pthread\_mutex\_lock(&node->mutex);

if (act != 0) {

std::cout << "Error:" << node->id - MIN\_PORT << ": Gateway mutex lock error\n",

node->active = false;

break;

}

std::vector <int> vectData = node->q.front();

node->q.pop();

act = pthread\_mutex\_unlock(&node->mutex);

if (act != 0){

std::cout << "Error:" << node->id - MIN\_PORT << ": Gateway mutex unlock error\n";

node->active = false;

break;

}

switch (vectData[0])

{

case CREATE:

{

zmq\_msg\_t msg;

act = zmq\_msg\_init\_size(&msg, 2 \* sizeof(int));

if (act == -1){

std::cout << "Error:" << node->id - MIN\_PORT << ": Message error\n";

break;

}

memcpy(zmq\_msg\_data(&msg), &vectData[0], 2 \* sizeof(int));

act = zmq\_msg\_send(&msg, req, 0);

if (act == -1){

std::cout << "Error:" << node->id - MIN\_PORT << ": Message error\n";

break;

}

int pid;

act = zmq\_recv(req, &pid, sizeof(int), 0);

if(act == -1){

std::cout << "Error:" << node->id - MIN\_PORT << ": Message error\n";

break;

}

if (vectData[1] < node->id){

node->L = new async\_node(vectData[1]);

} else {

node->R = new async\_node(vectData[1]);

}

std::cout << "Ok: " << pid << '\n';

zmq\_msg\_close(&msg);

break;

}

case EXEC:

{

zmq\_msg\_t msg;

int len = sizeof(int) \* 2;

act = zmq\_msg\_init\_size(&msg, len);

if (act == -1){

std::cout << "Error:" << node->id - MIN\_PORT << ": Message error\n";

break;

}

memcpy(zmq\_msg\_data(&msg), &vectData[0], len);

act = zmq\_msg\_send(&msg, req, 0);

if (act == -1){

std::cout << "Error:" << node->id - MIN\_PORT << ": Message error\n";

break;

}

long long ans;

act = zmq\_recv(req, &ans, sizeof(long long), 0);

if(act == -1){

std::cout << "Error:" << node->id - MIN\_PORT << ": Message error\n";

break;

}

if (ans >= 0 ) {

std::cout << "Ok:" << node->id - MIN\_PORT << ':' << ans << '\n';

} else {

std::cout << "Ok:" << node->id - MIN\_PORT << '\n';

}

zmq\_msg\_close(&msg);

break;

}

case REMOVE:

{

act = zmq\_send(req, &vectData[0], sizeof(int), 0);

if (act == -1){

std::cout << "Error:" << node->id - MIN\_PORT << ": Message error\n";

break;

}

int ans;

act = zmq\_recv(req, &ans, sizeof(int), 0);

if (act == -1){

std::cout << "Error:" << node->id - MIN\_PORT << ": Message error\n";

break;

}

break;

}

}

}

zmq\_close(req);

zmq\_ctx\_destroy(context);

delete node;

return NULL;

}

#endif

**computing.cpp**

#include <zmq.h>

#include <unistd.h>

#include <string.h>

#include <stack>

#include <ctime>

#include <iostream>

#include <vector>

const int CREATE = 1;

const int EXEC = 0;

const int REMOVE = -1;

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

{

if(argc != 2) {

std::cerr << "Not enough parameters" << std::endl;

exit(-1);

}

int act;

int id = atoi(argv[1]);

std::string port = "tcp://\*:" + std::to\_string(id);

void \*context = zmq\_ctx\_new();

void \*responder = zmq\_socket(context, ZMQ\_REP);

act = zmq\_bind(responder, port.c\_str());

if (act == -1) {

return -1;

}

std::stack <long long> stack;

while (true)

{

zmq\_msg\_t msg;

act = zmq\_msg\_init(&msg);

if (act == -1) {

return -1;

}

act = zmq\_msg\_recv(&msg, responder, 0);

if (act == -1) {

return -1;

}

int\* data = (int\*)zmq\_msg\_data(&msg);

int t = \*data;

switch (t)

{

case CREATE:

{

int n = \*(++data);

std::string id\_str = std::to\_string(n);

int pid = fork();

if (pid == -1){

return -1;

}

if (pid == 0){

execl("server", "server", id\_str.c\_str(), NULL);

} else {

act = zmq\_send(responder, &pid, sizeof(int), 0);

if (act == -1) {

return -1;

}

}

break;

}

case EXEC:

{

int commandNumber = \*(++data);

long long timer = 0;

if (commandNumber == 0){

long long start = std::time(nullptr);

stack.push(start);

timer = -1;

}

if ((commandNumber == 1) && !stack.empty()){

long long end = std::time(nullptr);

timer = (end - stack.top()) \* 1000;

}

if (commandNumber == 2){

stack.pop();

timer = -1;

}

act = zmq\_send(responder, &timer, sizeof(long long), 0);

if (act == -1) {

return -1;

}

break;

}

case REMOVE:

{

zmq\_send(responder, &id, sizeof(int), 0);

zmq\_close(responder);

zmq\_ctx\_destroy(context);

return 0;

}

}

act = zmq\_msg\_close(&msg);

if (act == -1) {

return -1;

}

}

}

**distribution.cpp**

#include "node.h"

int ConvertStrToNum(const std::string& commString){

if (commString == "start"){

return 0;

}

if (commString == "time"){

return 1;

}

return 2;

}

async\_node\* find\_node\_exec(async\_node\* ptr, int id)

{

if (ptr == nullptr)

return nullptr;

if (ptr->id > id)

return find\_node\_exec(ptr->L, id);

if (ptr->id < id)

return find\_node\_exec(ptr->R, id);

return ptr;

}

async\_node\* find\_node\_create(async\_node\* ptr, int id)

{

if (ptr == nullptr)

return nullptr;

if (ptr->L == nullptr && ptr->id > id)

return ptr;

if (ptr->R == nullptr && ptr->id < id)

return ptr;

if (ptr->id > id)

return find\_node\_create(ptr->L, id);

if (ptr->id < id)

return find\_node\_create(ptr->R, id);

return nullptr;

}

bool destroy\_node(async\_node\*& ptr, int id)

{

if (ptr == nullptr)

return false;

if (ptr->id > id)

return destroy\_node(ptr->L, id);

if (ptr->id < id)

return destroy\_node(ptr->R, id);

ptr->active = false;

ptr->make\_query({REMOVE});

if (ptr->L != nullptr)

destroy\_node(ptr->L, ptr->L->id);

if (ptr->R != nullptr)

destroy\_node(ptr->R, ptr->R->id);

ptr = nullptr;

return true;

}

bool ping(int id)

{

std::string port = protocol + std::to\_string(id);

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

void\* context = zmq\_ctx\_new();

void \*req = zmq\_socket(context, ZMQ\_REQ);

zmq\_socket\_monitor(req, ping.c\_str(), ZMQ\_EVENT\_CONNECTED | ZMQ\_EVENT\_CONNECT\_RETRIED);

void \*soc = zmq\_socket(context, ZMQ\_PAIR);

zmq\_connect(soc, ping.c\_str());

zmq\_connect(req, port.c\_str());

zmq\_msg\_t msg;

zmq\_msg\_init(&msg);

zmq\_msg\_recv(&msg, soc, 0);

uint8\_t\* data = (uint8\_t\*)zmq\_msg\_data(&msg);

uint16\_t event = \*(uint16\_t\*)(data);

zmq\_close(req);

zmq\_close(soc);

zmq\_msg\_close(&msg);

zmq\_ctx\_destroy(context);

return event % 2;

}

async\_node\* tree = nullptr;

int main()

{

while (true)

{

std::string command;

std::cin >> command;

if (command == "create")

{

int id;

std::cin >> id;

id += MIN\_PORT;

if (tree == nullptr)

{

std::string id\_str = std::to\_string(id);

int pid = fork();

if (pid == 0)

execl("server", "server", id\_str.c\_str(), NULL);

std::cout << "Ok: " << pid << '\n';

tree = new async\_node(id);

}

else

{

async\_node\* node = find\_node\_create(tree, id);

if (node != nullptr){

if (!ping(node->id))

{

std::cerr << "Error:" << id - MIN\_PORT << ": Parent is unavailable\n";

continue;

}

node->make\_query({CREATE, id});

}

else

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

}

}

if (command == "exec")

{

int id, commandNumber;

std::string commandString;

std::cin >> id >> commandString;

id += MIN\_PORT;

commandNumber = ConvertStrToNum(commandString);

std::vector <int> vectData(2);

vectData[0] = EXEC;

vectData[1] = commandNumber;

async\_node\* node = find\_node\_exec(tree, id);

if (node != nullptr){

node->make\_query(vectData);

if (!ping(id)) {

std::cerr << "Error:" << id - MIN\_PORT << ": Node is unavailable\n";

break;

}

} else {

std::cerr << "Error:" << id - MIN\_PORT << ": Not found\n";

}

}

if (command == "remove")

{

int id;

std::cin >> id;

id += MIN\_PORT;

if (!ping(id))

{

std::cerr << "Error:" << id - MIN\_PORT << ": Node is unavailable\n";

continue;

}

bool state = destroy\_node(tree, id);

if (state)

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

else

std::cerr << "Error: Not found\n";

}

if (command == "ping")

{

int id;

std::cin >> id;

id += MIN\_PORT;

async\_node\* node = find\_node\_exec(tree, id);

if (node != nullptr){

if (ping(id)) {

std::cout << "Ok: 1"<< '\n';

} else {

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

}

} else {

std::cerr << "Error:" << id - MIN\_PORT << ": Not found\n";

}

}

}

}

**Демонстрация работы программы**

anechka@po4ka:~/Documents/lab6-8/build-dir$ ./client create 4

Ok: 4256

create 5

Ok: 4264

ping 5

Ok: 1

ping 7

Error:7: Not found

remove 5

Ok

ping 5

Error:5: Not found

create 6

Ok: 4276

exec 6 start

Ok:6

exec 6 start

Ok:6

exexc 6 time

exec 6 time

Ok:6:11000

exec 6 stop

Ok:6

exec 6 time

Ok:6:35000

exec 6 stop

Ok:6

exec 6 time

Ok:6:0

create 7

Ok: 4299

sudo kill -9 4299

[sudo] password for anechka:

anechka@po4ka:~/Documents/lab6-8/build-dir$ lsof -i

COMMAND PID USER FD TYPE DEVICE SIZE/OFF NODE NAME

client 4251 anechka 9u IPv4 45050 0t0 TCP localhost:34630->localhost:1028 (ESTABLISHED)

client 4251 anechka 16u IPv4 46083 0t0 TCP localhost:58802->localhost:1030 (ESTABLISHED)

server 4256 anechka 9u IPv4 45047 0t0 TCP \*:1028 (LISTEN)

server 4256 anechka 10u IPv4 45051 0t0 TCP localhost:1028->localhost:34630 (ESTABLISHED)

server 4276 anechka 9u IPv4 45812 0t0 TCP \*:1030 (LISTEN)

server 4276 anechka 10u IPv4 46084 0t0 TCP localhost:1030->localhost:58802 (ESTABLISHED)

ping 7

Ok: 0

exec 7 start

Error:7: Node is unavailable

**Выводы**

В ходе лабораторной работы я научилась работать с библиотекой ZeroMQ. А также осуществила отложенные вычисления на нескольких вычислительных узлах.