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**

**Управлении серверами сообщений**

**Применение отложенных вычислений**

**Интеграция программных систем друг с другом**

Студент: Шаларь Игорь Павлович

Группа: М8О-208Б-20

Вариант: 5

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/IgShalar/OS/tree/main/os_lab6-8>

**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.

Топология: идеально сбалансированное бинарное дерево

Команда: сложение последовательности чисел

exec id n k1 ... kn

Тип проверки доступности узлов: pingall

**Общие сведения о программе**

Запуск:

\_$ cmake .

\_$ make

\_$ ./main

Написано для Unix.

**mylib.h:**

Вспомогательные функции. Например, работа с сокетами zeromq организована в виде классов.

Системные вызовы:

zmq\_ctx\_new() - создание контекста

zmq\_socket() - создание сокета

zmq\_connect() - подключение сокета к порту

zmq\_setsockopt() - настройка сокета

zmq\_close() - отключение сокета

zmq\_ctx\_destroy() - уничтожение контекста

zmq\_send() - отправка сообщения

zmq\_recv() - получение сообщения

execl() - заменяет текущий образ процесса новым образом процесса

**main.cpp:**

Управляющий узел.

Системные вызовыы

fork() - создание дочернего процесса

kill() - остановка процесса

**calc.cpp:**

вычислительный узел

**tree.h:**

Определяет куда добавить вершину

**Общий метод и алгоритм решения**

Каждый узел состоит из двух процессов, один передает выше команды снизу, другой слушает и передает ниже (если надо) команды сверху. При остановке узла, верхние узлы остаются работоспособными.

**Исходный код**

**CMakeLists.txt:**

cmake\_minimum\_required(VERSION 3.16)

set(CMAKE\_CXX\_STANDARD 14)

add\_executable(main main.cpp)

add\_executable(calc calc.cpp)

target\_link\_libraries(main zmq)

target\_link\_libraries(calc zmq)

**mylib.h:**

#include <zmq.h>

#include <unistd.h>

#include<iostream>

#include <sys/types.h>

#include <signal.h>

#include <chrono>

#include <thread>

using namespace std;

const int WAIT = 10;

const int LIM = 50;

int port = 48654;

class Request{

public:

Request(string s){

s = "tcp://localhost:" + s;

// cout << "Connecting request to " << s << endl;

ctx = zmq\_ctx\_new();

if (ctx == NULL){

perror("ctx");

}

req = zmq\_socket(ctx, ZMQ\_REQ);

if (req == NULL){

perror ("Can't create socket");

}

if (zmq\_connect (req, s.c\_str()) == -1) perror("Can't connect");

if (zmq\_setsockopt (req, ZMQ\_RCVTIMEO, &WAIT, sizeof(int)) == -1){

perror("Can't set wait");

}

if (zmq\_setsockopt (req, ZMQ\_LINGER, &WAIT, sizeof(int)) == -1){

perror("Can't set linger");

}

// cout << "Request to " << s << " connected." << endl;

}

~Request(){

if (zmq\_close(req) == -1) perror ("Can't close socket");

if (zmq\_ctx\_destroy(ctx) == -1) perror("Can't destroy context.");

}

string send(const string &s){

string res;

char ch[size];

zmq\_send (req, s.c\_str(), s.size() + 1, 0);

if (zmq\_recv (req, ch, size, 0) == -1) res = "-1";

else{

string temp(ch);

res = temp;

}

return res;

}

private:

const int size = LIM;

void \* req, \* ctx;

};

class Reply{

public:

Reply(string s){

s = "tcp://\*:" + s;

// cout << "Connecting reply to " << s << endl;

ctx = zmq\_ctx\_new();

if (ctx == NULL){

perror("ctx");

}

rep = zmq\_socket(ctx, ZMQ\_REP);

if (rep == NULL){

perror ("Can't create socket");

}

int rc = 0;

rc = zmq\_bind (rep, s.c\_str());

if (rc == -1) perror ("Connection failed");

// cout << "Reply to " << s << " connected." << endl;

}

~Reply(){

if (zmq\_close(rep) == -1) perror ("Can't close socket");

if (zmq\_ctx\_destroy(ctx) == -1) perror("Can't destroy context.");

}

string recieve(){

char ch[size];

zmq\_recv (rep, ch, size, 0);

string res(ch);

return res;

}

void reply(const string &s){

zmq\_send (rep, s.c\_str(), s.size() + 1, 0);

}

private:

const int size = LIM;

void \* rep, \* ctx;

};

void remove\_spaces (string &s){

int now = 0;

while (s[now] == ' ' && now < s.size()) now++;

s.erase(0, now);

}

string get\_word (string &s){

int now = 0;

while (s[now] == ' ' && now < s.size()) now++;

s.erase(0, now);

now = 0;

while (s[now] != ' ' && now < s.size()) now++;

string res = s.substr(0, now);

s.erase(0, now);

remove\_spaces(s);

return res;

}

string first\_word (const string &s){

int now = 0;

string res = "";

while (s[now] == ' ' && now < s.size()) now++;

while (s[now] != ' ' && now < s.size()){

res += s[now];

now++;

}

return res;

}

void create\_node(int my\_id, const string &p){ // add request port

string s = p;

s += " " + to\_string (my\_id);

// cout << "SEND\_STRING: " << s << endl;

int id = fork();

if (id == -1){

perror("Fork error");

}

else if (id == 0 && execl("calc", s.c\_str(), NULL) < 0){

perror("execl error");

}

}

string get\_port(){

if (port == 49152){

cout << "ERROR: Port limit exceeded." << endl;

return "ERROR";

}

string res = to\_string(port);

port++;

return res;

}

**tree.h:**

#include<set>

#include<iostream>

using namespace std;

struct node{

node \* l, \* r;

int v;

int s;

node(int a){

l = NULL;

r = NULL;

v = a;

s = 1;

}

};

struct Tree{

node \* root;

Tree(){

root = new node(-2);

}

void print(node \* now){

if (now == NULL) return;

cout << now->v << ": " << now->s << endl;

print(now->l);

print(now->r);

}

int cnt\_dfs(node \* now){

if (now == NULL) return 0;

int l = cnt\_dfs(now->l);

int r = cnt\_dfs(now->r);

now->s = l + r + 1;

return now->s;

}

void cnt(){

cnt\_dfs(root);

}

void del\_dfs(node \* now){

if (now == NULL) return;

del\_dfs(now->l);

del\_dfs(now->r);

delete now;

}

void add\_dfs(node \* now, int p, int val){

if (now == NULL) return;

if (now->v == p){

if (now->l == NULL) now->l = new node(val);

else if (now->r == NULL) now->r = new node (val);

return;

}

add\_dfs(now->l, p, val);

add\_dfs(now->r, p, val);

}

void add(int p, int val){

add\_dfs(root, p, val);

cnt();

}

void find\_dfs(node \* now, int &res, set <int> &s){

if (now == NULL) return;

if (now->l == NULL){

if (s.find(now->v) == s.end()) res = now->v;

return;

}

if (now->r == NULL){

if (s.find(now->v) == s.end()) res = now->v;

return;

}

if (now->l->s <= now->r->s) find\_dfs(now->l, res, s);

if (now->l->s >= now->r->s) find\_dfs(now->r, res, s);

}

int find(set <int> &s){

int res = -1;

find\_dfs(root, res, s);

return res;

}

~Tree(){

del\_dfs(root);

}

};

**main.cpp:**

#include"mylib.h"

#include<queue>

#include<set>

#include"tree.h"

using namespace std;

Tree tree;

set <int> nodes;

Request \* l = NULL, \* r = NULL;

set <int> ping (){

// this\_thread::sleep\_for(std::chrono::milliseconds(500 \* WAIT));

string m = "ping";

string s = "";

if (l != NULL) s += l->send(m);

s += " ";

if (r != NULL) s += r->send(m);

set <int> res = nodes;

remove\_spaces(s);

while (s != ""){

string t = get\_word(s);

int a = stoi(t);

if (a != -1) res.erase(a);

}

return res;

}

int main (){

string up\_port = get\_port();

string s = "";

int pr\_id = fork();

if (pr\_id == -1){

perror ("Fork error.");

exit(-1);

}

else if (pr\_id == 0){

Reply \* rep = new Reply(up\_port);

while(1){

string m = rep->recieve();

cout << m << endl;

rep->reply("GOT");

}

}

while (1){

cin >> s;

// cout << "INPUT: " << s << endl;

if (s == "exit"){

if (l != NULL) l->send("stop");

if (r != NULL) r->send("stop");

break;

}

else if (s == "create"){

int id;

string t;

cin >> id;

if (nodes.find(id) != nodes.end()){

cout << "Error: Already exists" << endl;

continue;

}

set <int> busy = ping();

int wh = tree.find(busy);

if (wh == -1){

cout << "Error: can't find avaliable parent" << endl;

continue;

}

nodes.insert(id);

tree.add(wh, id);

cout << "WHERE: " << wh << endl; //root has id = -2

if (wh == -2){

string temp = get\_port();

create\_node (id, get\_port() + " " + up\_port + " " + temp);

if (l == NULL) l = new Request(temp);

else r = new Request(temp);

}

else{

string m = to\_string(wh) + " c " + to\_string(id) + " " + get\_port() + " " + get\_port();

// cout << "TO SEND: " << m << endl;

if (l != NULL) l->send(m);

if (r != NULL) r->send(m);

}

}

else if (s == "exec"){

string t, res;

int n;

cin >> res;

res += " s";

cin >> n;

res += " " + to\_string(n);

for (int i = 0; i < n; i++){

cin >> t;

res += " " + t;

}

// cout << "TO SEND: " << res << endl;

string wh = first\_word(res);

int tr = stoi(wh);

if (nodes.find(tr) == nodes.end()){

cout << "Error:" << tr << ": Not found" << endl;

continue;

}

set <int> qw = ping();

if (qw.find(tr) != qw.end()){

cout << "Error:" << tr << ": Node is unavailable" << endl;

continue;

}

if (l != NULL) l->send(res);

if (r != NULL) r->send(res);

}

else if (s == "pingall"){

set<int> temp = ping();

cout << "OK: ";

if (temp.empty()) cout << "-1";

else for (auto i : temp) cout << i << "; ";

cout << endl;

}

else if (s == "kill"){

int w;

cin >> w;

string m = to\_string(w) + " k";

if (l != NULL) l->send(m);

if (r != NULL) r->send(m);

}

}

if (kill(pr\_id, SIGKILL) == -1) perror ("Can't kill");

if (l != NULL) delete l;

if (r != NULL) delete r;

return 0;

}

**calc.cpp:**

#include"mylib.h"

using namespace std;

int main (int a, char \* tr[]){

string qw (\* tr);

// cout << "GOT\_STRING: " << qw << endl;

string up\_port = get\_word(qw);

string req\_port = get\_word(qw);

string rep\_port = get\_word(qw);

string my\_id = get\_word(qw);

Request \* p = new Request(req\_port), \* l = NULL, \* r = NULL;

Reply \* prep = new Reply(rep\_port), \* rep = NULL;

int pr\_id = fork();

if (pr\_id == -1){

perror ("Fork error.");

exit(-1);

}

else if (pr\_id == 0){

Request \* p\_sec = new Request(req\_port);

rep = new Reply(up\_port);

while(1){

string m = rep->recieve();

rep->reply("OK");

p\_sec->send(m);

// cout << my\_id << ": up " << m << endl;

}

}

// cout << my\_id << ": started" << endl;

p->send("OK: " + to\_string(getpid()) + " " + to\_string(pr\_id));

while (1){

string s = prep->recieve();

if (first\_word(s) == "ping"){

string t1 = "-1";

string t2 = "-1";

if (l != NULL) t1 = l->send(s);

if (r != NULL) t2 = r->send(s);

if (t1 != "-1") t1 += " " + my\_id;

else t1 = my\_id;

if (t2 != "-1") t1 += " " + t2;

prep->reply(t1);

}

prep->reply("OK");

if (first\_word(s) == "stop"){

if (l != NULL) l->send(s);

if (r != NULL) r->send(s);

break;

}

else if (first\_word(s) != my\_id){

if (l != NULL) l->send(s);

if (r != NULL) r->send(s);

}

else{

get\_word(s);

string t = get\_word(s);

if (t == "c"){

int new\_id = stoi(get\_word(s));

string up = get\_word(s);

string rep = get\_word(s);

create\_node(new\_id, up + " " + up\_port + " " + rep);

if (l == NULL) l = new Request(rep);

else r = new Request(rep);

// cout << my\_id << ": new node id = " << new\_id << endl;

}

else if (t == "s"){

long long n, a, res = 0;

n = stoi(get\_word(s));

for (int i = 0; i < n; i++){

a = stoi(get\_word(s));

res += a;

}

// cout << my\_id << ": SOLVED" << endl;

string result = "OK:" + my\_id + ": " + to\_string(res);

p->send(result);

//cout << my\_id << ": " << res << endl;

}

else if (t == "k"){

if (l != NULL) l->send("stop");

if (r != NULL) r->send("stop");

break;

}

}

}

if (kill(pr\_id, SIGKILL) == -1) perror ("Can't kill");

if (p != NULL) delete p;

delete prep;

if (l != NULL) delete l;

if (rep != NULL) delete rep;

if (r != NULL) delete r;

cout << my\_id << ": STOPPED" << endl;

return 0;

}

**Демонстрация работы программы**

igor@igor-VirtualBox:/media/sf\_VM\_Shared$ ./main

create 1

WHERE: -2

OK: 2294 2301

create 2

WHERE: -2

OK: 2306 2313

create 3

WHERE: 2

OK: 2318 2325

create 4

WHERE: 1

OK: 2330 2337

create 5

WHERE: 2

OK: 2342 2349

create 6

WHERE: 1

OK: 2354 2361

exec 6 2 3 4

OK:6: 7

kill 2

5: STOPPED

3: STOPPED

2: STOPPED

pingall

OK: 2; 3; 5;

exec 1 2 3 4

OK:1: 7

kill 6

6: STOPPED

pingall

OK: 2; 3; 5; 6;

exit

4: STOPPED

1: STOPPED

**Выводы**

Познакомился с серверами сообщений, отложенными вычислениями. Получил ценный опыт использования этих технологий для реализации распределенной системы по асинхронной обработке запросов.