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»**

**Управлении серверами сообщений. Применение отложенных вычислений. Интеграция программных систем друг с другом.**

Студент: Попов Илья Павлович

Группа: М80-206Б-20

Вариант: 35

Преподаватель: Соколов Андрей Алексеевич

Дата: 25.12.2021

Оценка: 5

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2021

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.

Управляющий узел отвечает за ввод команд от пользователя и отправку этих команд на вычислительные узлы. Список основных поддерживаемых команд:

Создание нового вычислительного узла

Формат команды: create id [parent]

id – целочисленный идентификатор нового вычислительного узла

parent – целочисленный идентификатор родительского узла. Если топологией не предусмотрено введение данного параметра, то его необходимо игнорировать (если его ввели)

Формат вывода: «Ok: pid», где pid – идентификатор процесса для созданного вычислительного узла

«Error: Already exists» - вычислительный узел с таким идентификатором уже существует

«Error: Parent not found» - нет такого родительского узла с таким идентификатором

«Error: Parent is unavailable» - родительский узел существует, но по каким-то причинам с ним не удается связаться

«Error: [Custom error]» - любая другая обрабатываемая ошибка

Пример:

> create 10 5 Ok: 3128

Примечания: создание нового управляющего узла осуществляется пользователем программы при помощи запуска исполняемого файла. Id и pid — это разные идентификаторы.

Исполнение команды на вычислительном узле

Формат команды: exec id [params]

id – целочисленный идентификатор вычислительного узла, на который отправляется команда Формат вывода:

«Ok:id: [result]», где result – результат выполненной команды

«Error:id: Not found» - вычислительный узел с таким идентификатором не найден

«Error:id: Node is unavailable» - по каким-то причинам не удается связаться с вычислительным узлом

«Error:id: [Custom error]» - любая другая обрабатываемая ошибка

Пример: Можно найти в описании конкретной команды, определенной вариантом задания.

Примечание: выполнение команд должно быть асинхронным. Т.е. пока выполняется команда на одном из вычислительных узлов, то можно отправить следующую команду на другой вычислительный узел.

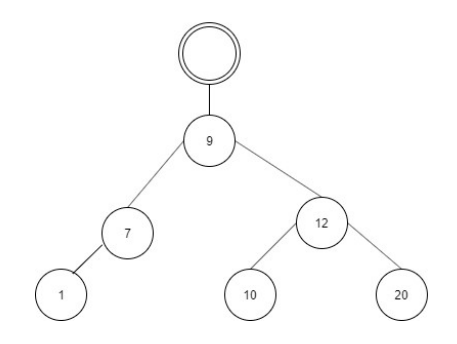
**Вариант №35**

Топология – 4

Тип команды – 1

Тип проверки доступности узлов - 2

**Топология 3**



Все вычислительные узлы хранятся в бинарном дереве поиска. [parent] — является необязательным параметром.

**Топология 4**

Аналогично топологии 3, но узлы находятся в идеально сбалансированном бинарном дереве. Каждый следующий узел должен добавляться в самое наименьшее поддерево.

**Набор команд 1** (подсчет суммы n чисел)

Формат команды: exec id n k1 … kn id – целочисленный идентификатор вычислительного узла, на который отправляется команда n – количество складываемых чисел (от 1 до 108) k1 … kn – складываемые числа

Пример:

> exec 10 3 1 2 3 Ok:10: 6

**Тип проверки доступности узлов**

Команда проверки 2

Формат команды: ping id

Команда проверяет доступность конкретного узла. Если узла нет, то необходимо выводить ошибку: «Error: Not found»

Пример: > ping 10

Ok: 1 // узел 10 доступен

> ping 17

Ok: 0 // узел 17 недоступен

Листинг программы

BalancedTree.h

#ifndef BALANCED\_TREE\_H

#define BALANCED\_TREE\_H

#include <bits/stdc++.h>

class BalancedTree {

class BalancedTreeNode {

public:

int id;

BalancedTreeNode\* left;

BalancedTreeNode\* right;

int height;

bool available;

BalancedTreeNode (int id) {

this->id = id;

available = true;

left = NULL;

right = NULL;

}

void CheckAvailability (int id) {

if (this->id == id){

available = false;

}

else {

if (left != NULL) {

left->CheckAvailability(id);

}

if (right != NULL) {

right->CheckAvailability(id);

}

}

}

void Remove (int id, set<int> &ids) {

if (left != NULL && left->id == id) {

left->RecursionRemove(ids);

ids.erase(left->id);

delete left;

left = NULL;

}

else if (right != NULL && right->id == id) {

right->RecursionRemove(ids);

ids.erase(right->id);

delete right;

right = NULL;

}

else {

if (left != NULL) {

left->Remove(id, ids);

}

if (right != NULL) {

right->Remove(id, ids);

}

}

}

void RecursionRemove (set<int> &ids) {

if (left != NULL) {

left->RecursionRemove(ids);

ids.erase(left->id);

delete left;

left = NULL;

}

if (right != NULL) {

right->RecursionRemove(ids);

ids.erase(right->id);

delete right;

right = NULL;

}

}

void AddInNode (int id, int parent\_id, set<int> &ids) {

if (this->id == parent\_id) {

if (left == NULL){

left = new BalancedTreeNode(id);

}

else {

right = new BalancedTreeNode(id);

}

ids.insert(id);

}

else {

if (left != NULL) {

left->AddInNode(id, parent\_id, ids);

}

if (right != nullptr) {

right->AddInNode(id, parent\_id, ids);

}

}

}

int MinimalHeight() {

if (left == NULL || right == NULL) {

return 0;

}

int left\_height = -1;

int right\_height = -1;

if (left != NULL && left->available == true) {

left\_height = left->MinimalHeight();

}

if (right != NULL && right->available == true) {

right\_height = right->MinimalHeight();

}

if (right\_height == -1 && left\_height == -1) {

available = false;

return -1;

}

else if (right\_height == -1) {

return left\_height + 1;

}

else if (left\_height == -1) {

return right\_height + 1;

}

else {

return min(left\_height, right\_height) + 1;

}

}

int IDMinimalHeight(int height, int current\_height) {

if (height < current\_height) {

return -2;

}

else if (height > current\_height) {

int current\_id = -2;

if (left != NULL && left->available == true) {

current\_id = left->IDMinimalHeight(height, (current\_height + 1));

}

if (right != NULL && right->available == true && current\_id == -2){

current\_id = right->IDMinimalHeight(height, (current\_height + 1));

}

return current\_id;

}

else {

if (left == NULL || right == NULL){

return id;

}

return -2;

}

}

~BalancedTreeNode() {}

};

private:

BalancedTreeNode\* root;

public:

set<int> ids;

BalancedTree() {

root = new BalancedTreeNode(-1);

}

bool Exist(int id) {

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

return true;

}

return false;

}

void AvailabilityCheck(int id) {

root->CheckAvailability(id);

}

int FindID() {

int h = root->MinimalHeight();

return root->IDMinimalHeight(h, 0);

}

void AddInTree(int id, int parent) {

root->AddInNode(id, parent, ids);

}

void RemoveFromRoot(int idElem) {

root->Remove(idElem, ids);

}

~BalancedTree() {

root->RecursionRemove(ids);

delete root;

}

};

#endif

CalculationNode.h

#include <bits/stdc++.h>

#include "ZMQFunctions.h"

#include "unistd.h"

class CalculationNode {

private:

zmq:: context\_t context;

public:

zmq:: socket\_t left, right, parent;

int left\_port, right\_port, parent\_port;

int id, left\_id = -2, right\_id = -2, parent\_id;

CalculationNode(int id, int parent\_port, int parent\_id):

id(id),

parent\_port(parent\_port),

parent\_id(parent\_id),

left(context, ZMQ\_REQ),

right(context, ZMQ\_REQ),

parent(context, ZMQ\_REP)

{

if (id != -1) {

connect(parent, parent\_port);

}

}

string create (int child\_id) {

int port;

bool isleft = false;

if (left\_id == -2) {

left\_port = bind(left, child\_id);

left\_id = child\_id;

port = left\_port;

isleft = true;

}

else if (right\_id == -2) {

right\_port = bind(right, child\_id);

right\_id = child\_id;

port = right\_port;

}

else {

string fail = "Error: can not create the calculation node";

return fail;

}

int fork\_id = fork();

if (fork\_id == 0) {

if (execl("./client", "client", to\_string(child\_id).c\_str(), to\_string(port).c\_str(), to\_string(id).c\_str(), (char\*)NULL) == -1) {

cout << "Error: can not run the execl-command" << endl;

exit(EXIT\_FAILURE);

}

}

else {

string child\_pid;

try {

if (isleft) {

left.setsockopt(ZMQ\_SNDTIMEO, 3000);

send\_message(left, "pid");

child\_pid = receive\_message(left);

}

else {

right.setsockopt(ZMQ\_SNDTIMEO, 3000);

send\_message(right, "pid");

child\_pid = receive\_message(right);

}

return "Ok: " + child\_pid;

}

catch (int) {

string fail = "Error: can not connect to the child";

return fail;

}

}

}

string ping (int id) {

string answer = "Ok: 0";

if (this->id == id) {

answer = "Ok: 1";

return answer;

}

else if (left\_id == id) {

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

send\_message(left, message);

try {

message = receive\_message(left);

if (message == "Ok: 1") {

answer = message;

}

}

catch(int){}

}

else if (right\_id == id) {

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

send\_message(right, message);

try {

message = receive\_message(right);

if (message == "Ok: 1") {

answer = message;

}

}

catch(int){}

}

return answer;

}

string sendstring (string string, int id) {

std::string answer = "Error: Parent not found";

if (left\_id == -2 && right\_id == -2) {

return answer;

}

else if (left\_id == id) {

if (ping(left\_id) == "Ok: 1") {

send\_message(left, string);

try{

answer = receive\_message(left);

}

catch(int){}

}

}

else if (right\_id == id) {

if (ping(right\_id) == "Ok: 1") {

send\_message(right, string);

try {

answer = receive\_message(right);

}

catch(int){}

}

}

else {

if (ping(left\_id) == "Ok: 1") {

std::string message = "send " + to\_string(id) + " " + string;

send\_message(left, message);

try {

message = receive\_message(left);

}

catch(int) {

message = "Error: Parent not found";

}

if (message != "Error: Parent not found") {

answer = message;

}

}

if (ping(right\_id) == "Ok: 1") {

std::string message = "send " + to\_string(id) + " " + string;

send\_message(right, message);

try {

message = receive\_message(right);

}

catch(int) {

message = "Error: Parent not found";

}

if (message != "Error: Parent not found") {

answer = message;

}

}

}

return answer;

}

string exec ( string string) {

istringstream string\_thread(string);

int result = 0;

int amount, number;

string\_thread >> amount;

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

string\_thread >> number;

result += number;

}

std::string answer = "Ok: " + to\_string(id) + ": " + to\_string(result);

return answer;

}

string kill () {

if (left\_id != -2){

if (ping(left\_id) == "Ok: 1") {

std::string message = "kill";

send\_message(left, message);

try {

message = receive\_message(left);

}

catch(int){}

unbind(left, left\_port);

left.close();

}

}

if (right\_id != -2) {

if (ping(right\_id) == "Ok: 1") {

std::string message = "kill";

send\_message(right, message);

try {

message = receive\_message(right);

}

catch (int){}

unbind(right, right\_port);

right.close();

}

}

return to\_string(parent\_id);

}

~CalculationNode() {}

};

Client.cpp

#include <bits/stdc++.h>

#include "CalculationNode.h"

#include "ZMQFunctions.h"

#include "BalancedTree.h"

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

CalculationNode node(atoi(argv[1]), atoi(argv[2]), atoi(argv[3]));

while(true) {

string message;

string command;

message = receive\_message(node.parent);

istringstream request(message);

request >> command;

if (command == "pid") {

string answer = to\_string(getpid());

send\_message(node.parent, answer);

}

else if (command == "ping") {

int child;

request >> child;

string answer = node.ping(child);

send\_message(node.parent, answer);

}

else if (command == "create") {

int child;

request >> child;

string answer = node.create(child);

send\_message(node.parent, answer);

}

else if (command == "exec") {

string str;

getline(request, str);

string answer = node.exec(str);

send\_message(node.parent, answer);

}

else if (command == "kill") {

string answer = node.kill();

send\_message(node.parent, answer);

disconnect(node.parent, node.parent\_port);

node.parent.close();

break;

}

}

return 0;

}

Makefile

files: server client

server: Server.cpp

g++ -fsanitize=address Server.cpp -lzmq -o server -w

client: Client.cpp

g++ -fsanitize=address Client.cpp -lzmq -o client -w

clean:

rm -rf server client

Server.cpp

#include <bits/stdc++.h>

#include "CalculationNode.h"

#include "ZMQFunctions.h"

#include "BalancedTree.h"

void menu(){

cout << "Avaliable commands:\n";

cout << "1. create <id>\n";

cout << "2. exec <id> <n> <n1 n2... n>\n";

cout << "3. ping <id>\n";

cout << "4. kill <id>\n";

}

int main() {

string command;

CalculationNode node(-1, -1, -1);

string answer;

BalancedTree tree;

menu();

while (( cout << "> ") && ( cin >> command)) {

if (command == "create") {

int child; cin >> child;

if (tree.Exist(child)) {

cout << "Error: Already exists\n";

}

else {

while (true) {

int idParent = tree.FindID();

if (idParent == node.id) {

answer = node.create(child);

tree.AddInTree(child, idParent);

break;

}

else {

string message = "create " + to\_string(child);

answer = node.sendstring(message, idParent);

if (answer == "Error: Parent not found") {

tree.AvailabilityCheck(idParent);

}

else {

tree.AddInTree(child, idParent);

break;

}

}

}

cout << answer << endl;

}

}

else if (command == "exec") {

if (!tree.Exist(child)) {

cout << "Error: Node does not exist!\n";

}

string str;

int child; cin >> child;

getline(cin, str);

if (!tree.Exist(child)) {

cout << "Error: Parent is not existed\n";

}

else {

string message = "exec " + str;

answer = node.sendstring(message, child);

cout << answer << endl;

}

}

else if (command == "ping") {

int child; cin >> child;

if (!tree.Exist(child)) {

cout << "Ok: 0\n";

}

else if (node.left\_id == child || node.right\_id == child) {

answer = node.ping(child);

cout << answer << endl;

}

else {

string message = "ping " + to\_string(child);

answer = node.sendstring(message, child);

cout << answer << endl;

}

}

else if (command == "kill") {

int child; cin >> child;

string message = "kill";

if (!tree.Exist(child)) {

cout << "Error: Parent is not existed\n";

}

else {

answer = node.sendstring(message, child);

if (answer != "Error: Parent not found") {

tree.RemoveFromRoot(child);

if (child == node.left\_id){

node.left\_id = -2;

unbind(node.left, node.left\_port);

answer = "Ok";

}

else if (child == node.right\_id) {

node.right\_id = -2;

unbind(node.right, node.right\_port);

answer = "Ok";

}

else {

message = "clear " + to\_string(child);

answer = node.sendstring(message, stoi(answer));

}

cout << answer << endl;

}

}

}

else {

cout << "Please enter correct command!\n\n";

menu();

}

}

node.kill();

return 0;

}

ZMQFunctions.h

#ifndef ZMQ\_H

#define ZMQ\_H

#include <bits/stdc++.h>

#include <zmq.hpp>

const int MAIN\_PORT = 4040;

using namespace std;

void send\_message(zmq::socket\_t &socket, const string &msg) {

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

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

socket.send(message);

}

string receive\_message(zmq::socket\_t &socket) {

zmq::message\_t message;

int chars\_read;

try {

chars\_read = (int)socket.recv(&message);

}

catch (...) {

chars\_read = 0;

}

if (chars\_read == 0) {

throw -1;

}

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

return received\_msg;

}

void connect(zmq::socket\_t &socket, int port) {

string address = "tcp://127.0.0.1:" + to\_string(port);

socket.connect(address);

}

void disconnect(zmq::socket\_t &socket, int port) {

string address = "tcp://127.0.0.1:" + to\_string(port);

socket.disconnect(address);

}

int bind(zmq::socket\_t &socket, int id) {

int port = MAIN\_PORT + id;

string address = "tcp://127.0.0.1:" + to\_string(port);

while(1){

try{

socket.bind(address);

break;

}

catch(...){

port++;

}

}

return port;

}

void unbind(zmq::socket\_t &socket, int port) {

string address = "tcp://127.0.0.1:" + to\_string(port);

socket.unbind(address);

}

#endif

Примеры работы

lunidep@lunidep-VirtualBox:~/Desktop/OS/lab6$ ./server

Avaliable commands:

1. create <id>

2. exec <id> <n> <n1 n2... n>

3. ping <id>

4. kill <id>

> create 5

Ok: 7746

> create 2

Ok: 7771

> create 7

Ok: 7774

> ping 2

Ok: 1

> kill 2

Ok

> ping 2

Ok: 0

> exec 7 3 1 2 3

Ok: 7: 6

Вывод

В процессе выполнения данной лабораторной работы мною была реализована распределенная система по асинхронной обработке запросов. В моей программе использовался протокол передачи данных tcp, в котором, в отличие от ipc общение между процессами происходит через определенные порты, а не через временные файлы.

Обмен сообщений происходит посредством функций библиотеки ZMQ, а в частности, ее паттерном «Request – Reply». Это один из самых простых и прямолинейных паттернов, который своей реализацией очень напоминает pipe. Материала для реализации данной лабораторной работы потребовалось довольно много и я получил полезный опыт изучения англоязычной документации.

Также хорошей тренировкой стала реализация идеально сбалансированного бинарного дерева на С++.