# Московский Авиационный Институт (Национальный Исследовательский Университет)

Институт №8 “Компьютерные науки и прикладная математика” Кафедра №806 “Вычислительная математика и программирование”

## Лабораторная работа №5-7 по курсу

**«Операционные системы»**

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

Группа: М80-210Б-22

Студент: Казарцев Денис Вячеславович

Вариант:33

Преподаватель: Соколов А.А. Оценка: Дата: Подпись:

Москва, 2023

# 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.

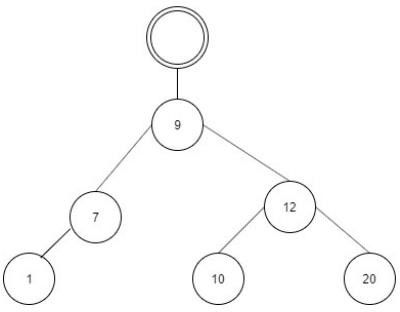
Примечание: выполнение команд должно быть асинхронным. Т.е. пока выполняется команда на одном из вычислительных узлов, то можно отправить следующую команду на другой вычислительный узел.

**Вариант №33**

Топология – 4 Тип команды – 1

Тип проверки доступности узлов - 1

**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 10^8) k1 … kn – складываемые числа

Пример:

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

## Тип проверки доступности узлов

Команда проверки 1 Формат команды: pingall

Вывод всех недоступных узлов вывести разделенные через точку запятую.

Пример: > pingall Ok: -1

// Все узлы доступны > pingall Ok: 7;10;15

// узлы 7, 10, 15 — недоступны

# Листинг программы

**AVL-tree.hpp** #include <iostream> #include <set>

class Node{

public:

int id;

int height; Node\* left; Node\* right;

Node(int new\_id){ id = new\_id;

left = right = nullptr; height = 1;

}

};

class AVL\_tree { private:

Node \*root;

void destroy\_node(Node \* node){ if(node != nullptr){

destroy\_node(node->left); destroy\_node(node->right); delete node;

}

}

void print\_tree(Node \*node) { if (node != nullptr) {

print\_tree(node->left);

for(int i = 0; i < node->height; i++){ std::cout << " ";

}

std::cout << node->id << "\n"; print\_tree(node->right);

}

}

void fix\_height(Node\* node){

if(node->left != nullptr && node->right != nullptr){

node->height = std::max(node->left->height, node->right->height) + 1;

} else if (node->left != nullptr){

node->height = node->left->height + 1;

} else if (node->right != nullptr){

node->height = node->right->height + 1;

} else{

node->height = 1;

}

}

int bfactor(Node \* node){

if(node->left != nullptr && node->right != nullptr){ return node->right->height - node->left->height;

} else if (node->left != nullptr){ return -node->left->height;

} else if (node->right != nullptr){ return node->right->height;

} else{

return 0;

}

}

Node\* rotateright(Node \* node){ Node\* new\_node = node->left; node->left = new\_node->right; new\_node->right = node; fix\_height(node); fix\_height(new\_node);

return new\_node;

}

Node\* rotateleft(Node \* node){ Node\* q = node->right;

node->right = q->left; q->left = node; fix\_height(node); fix\_height(q);

return q;

}

Node\* balance(Node\* node){ fix\_height(node);

if(bfactor(node) == 2){

if(bfactor(node->right) < 0){

node->right = rotateright(node->right);

}

return rotateleft(node);

}

if(bfactor(node) == -2){

if(bfactor(node->left) > 0){

node->left = rotateleft(node->left);

}

return rotateright(node);

}

return node;

}

Node\* insert\_(Node\* p, int id){ if(p == nullptr){

return new Node(id);

}

if(id < p->id){

p->left = insert\_(p->left, id);

} else {

p->right = insert\_(p->right, id);

}

return balance(p);

}

Node \* remove(Node \*node, int id){ if (node == nullptr){

return node;

}

if (id == node->id){ free(node);

return nullptr;

}

if (id < node->id){

node->left = remove(node->left, id);

} else {

node->right = remove(node->right, id);

}

return balance(node);

}

public:

std::vector<int> all\_elem; AVL\_tree(){

root = nullptr;

}

~AVL\_tree(){ destroy\_node(root);

}

void insert(int id){ all\_elem.push\_back(id); root = insert\_(root, id);

}

void print(){

this->print\_tree(root);

}

Node \* get\_root(){ return root;

}

Node \*find(int id) { Node \*node = root;

while (node != nullptr && node->id != id) { if (node->id > id){

node = node->left;

} else{

node = node->right;

}

}

return node;

}

void remove\_t(int id){ root = remove(root, id);

}

};

### Socet.hpp

#include <iostream> #include "zmq.h" #include <cstring> #include <vector>

const int MAIN\_id = 4000;

const char \* CLIENT\_NODE = "cl"; void \* context = zmq\_ctx\_new();

enum actions{ fail = 0,

success = 1,

create = 2,

pingall = 3,

ping = 4,

exec = 5,

destroy = 6,

remove\_t = 7

};

struct zmqmessage{ actions act;

int perant; int id;

};

class Socket{ public:

void \* socket; int id;

Socket(){ id = -1;

socket = zmq\_socket(context, ZMQ\_PAIR);

}

void bind(int node){ id = node;

if(zmq\_bind(socket, ("tcp://\*:" + std::to\_string(MAIN\_id + id)).c\_str())){ throw std::runtime\_error("ZMQ\_bind");

}

}

void unbind(){ if (id == -1){

return;

}

if(zmq\_unbind(socket, ("tcp://\*:" + std::to\_string(MAIN\_id + id)).c\_str())){ throw std::runtime\_error("ZMQ\_unbind" + std::to\_string(id));

}

id = -1;

}

void connect(int node){ id = node;

if(zmq\_connect(socket, ("tcp://localhost:" + std::to\_string(MAIN\_id + id)).c\_str())){ throw std::runtime\_error("ZMQ\_con");

}

}

void disconnect(){ if (id == -1){

return;

}

if(zmq\_disconnect(socket, ("tcp://\*:" + std::to\_string(MAIN\_id + id)).c\_str())){ throw std::runtime\_error("ZMQ\_discon");

}

id = -1;

}

void send\_message(const zmqmessage \* mes){ zmq\_send(socket, mes, sizeof(zmqmessage), 0);

}

void recive\_message(zmqmessage \*&mes){ zmq\_recv(socket, mes, sizeof(zmqmessage), 0);

}

void close(){ zmq\_close(socket);

}

void new\_socket(){ id = -1;

socket = zmq\_socket(context, ZMQ\_PAIR);

}

};

### Control.cpp

#include "soket.h" #include "AVL-tree.hpp"

pid\_t create\_process() { pid\_t pid = fork();

if (pid == -1) {

throw std::runtime\_error("fork error!\n");

}

return pid;

}

int main(){ Socket socket; AVL\_tree tree; std::string s; int id;

while(std::cin >> s){ if (s == "create"){

std::cin >> id;

Node \*node = tree.find(id);

if (node != nullptr) {

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

}

if (tree.get\_root() == nullptr){

tree.insert(id); socket.bind(id);

pid\_t pid = create\_process(); if (pid == 0){

execl(CLIENT\_NODE, CLIENT\_NODE, std::to\_string(id).c\_str(), nullptr);

}

} else {

zmqmessage \* mes = new zmqmessage({create, 0, id}); socket.send\_message(mes);

tree.insert(id); delete mes;

}

}

else if (s == "remove"){ std::cin >> id;

if (tree.find(id) == nullptr){

std::cout << "Error: "<< id <<": Not found" << "\n"; continue;

}

if (id == tree.get\_root()->id){ tree.remove\_t(id);

zmqmessage \* mes = new zmqmessage({destroy, 0, 0}); socket.send\_message(mes);

socket.new\_socket(); delete mes; continue;

}

tree.remove\_t(id);

zmqmessage \* mes = new zmqmessage({remove\_t, 0, id}); socket.send\_message(mes);

delete mes;

}

else if (s == "ping"){

if (tree.get\_root() == nullptr){ continue;

}

zmqmessage \* mes = new zmqmessage({ping, 0, 0}); socket.send\_message(mes);

delete mes;

}

else if (s == "pingall"){

zmqmessage \* mes = new zmqmessage({pingall, 0, 0}); socket.send\_message(mes); socket.recive\_message(mes);

std::set<int> fork\_ids; while(mes->perant != 1){

fork\_ids.insert(mes->id); socket.recive\_message(mes);

}

fork\_ids.insert(mes->id); bool flag = false; std::cout << "Ok:";

for(int i = 0; i < tree.all\_elem.size(); i++){ if(fork\_ids.count(tree.all\_elem[i]) == 0){

flag = true;

std::cout << tree.all\_elem[i] << ";";

}

}

if (flag == false){ std::cout << "-1";

}

std::cout << "\n"; delete mes;

}

else if (s == "exec"){ int n;

std::cin >> n; if(tree.find(n) == nullptr){

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

}

zmqmessage \* mes = new zmqmessage({exec, 0, n}); socket.send\_message(mes); socket.recive\_message(mes);

std::cout << "Ok:" << n <<":" << mes->id << "\n"; delete mes;

}

}

tree.~AVL\_tree(); socket.close(); zmq\_ctx\_destroy(context);

}

### Client.cpp

#include "soket.h"

pid\_t create\_process() { pid\_t pid = fork();

if (pid == -1) {

throw std::runtime\_error("fork error!\n");

}

return pid;

}

void make\_node(Socket &child, int id){ child.bind(id);

pid\_t fork\_id = create\_process(); if (fork\_id == 0) {

execl(CLIENT\_NODE, CLIENT\_NODE, std::to\_string(id).c\_str(), nullptr);

}

}

int main(int argc, char \*\* argv){ if (argc != 2){

throw std::logic\_error("./NAME\_PROGRAMM id");

}

int node\_id = std::atoi(argv[1]); Socket socket; socket.connect(node\_id); Socket left\_child, right\_child; zmqmessage \* command; zmqmessage \* ans\_left; zmqmessage \* ans\_right;

std::cout << "Ok " << getpid() << "\n";

while(true){ socket.recive\_message(command); if (command->act == create){

if(command->id < node\_id && left\_child.id == -1){ make\_node(left\_child, command->id);

}

else if(command->id > node\_id && right\_child.id == -1){ make\_node(right\_child, command->id);

}

else if (command->id < node\_id && left\_child.id != -1){ left\_child.send\_message(command);

} else {

right\_child.send\_message(command);

}

} else if(command->act == exec) { if (node\_id == command->id){

int n; std::cin >> n;

long long res = 0; int elem;

for(int i = 0; i < n; i++){ std::cin >> elem; res += elem;

}

command->id = res;

} else if (node\_id < command->id){ right\_child.send\_message(command); right\_child.recive\_message(command);

} else {

left\_child.send\_message(command); left\_child.recive\_message(command);

}

socket.send\_message(command);

}

else if(command->act == ping){ std::cout << "\n \n";

std::cout << "IM " << node\_id << "\n";

std::cout << "LEFT CHILD " << left\_child.id << "\n"; std::cout << "RIGHT CHILD " << right\_child.id << "\n"; std::cout << " \n";

if(left\_child.id != -1){ left\_child.send\_message(command);

}

if(right\_child.id != -1){ right\_child.send\_message(command);

}

}

else if (command->act == pingall){ command->perant += 1;

zmqmessage \* ans = new zmqmessage({fail, command->perant, node\_id}); if(left\_child.id != -1){

ans->act = pingall; left\_child.send\_message(command); left\_child.recive\_message(ans\_left); while(ans\_left->perant - command->perant > 1){

socket.send\_message(ans\_left); left\_child.recive\_message(ans\_left);

}

socket.send\_message(ans\_left);

}

if(right\_child.id != -1){ ans->act = pingall;

right\_child.send\_message(command); right\_child.recive\_message(ans\_right); while(ans\_right->perant - command->perant > 1){

socket.send\_message(ans\_right); right\_child.recive\_message(ans\_right);

}

socket.send\_message(ans\_right);

}

socket.send\_message(ans); delete ans;

} else if (command->act == destroy){ if(left\_child.id != -1){

left\_child.send\_message(command);

}

if(right\_child.id != -1){ right\_child.send\_message(command);

}

break;

} else if (command->act == remove\_t){ if (command->id == left\_child.id){

command->act = destroy; left\_child.send\_message(command); left\_child.close(); left\_child.new\_socket();

} else if (command->id == right\_child.id) { command->act = destroy; right\_child.send\_message(command); right\_child.new\_socket();

} else if (command->id > node\_id){ right\_child.send\_message(command);

} else{

left\_child.send\_message(command);

}

command->act = fail;

}

}

left\_child.close(); right\_child.close();

}

# Примеры работы

Ananasik558@ Ananasik558:~/Desktop/lab\_os/lab5-6$ ./con create 10

Ok 8064

create 20 Killed

Ananasik558@ Ananasik558:~/Desktop/lab\_os/lab5-6$ ./con

[1]+ Killed ./con (wd: ~/Desktop/lab\_os/lab5-6/build) (wd now: ~/Desktop/lab\_os/lab5-6)

create 20

Ok 8411

create 30

Ok 8418

create 10

Ok 8424

pingall Ok:-1

exec 10

3 2 3 4

Ok:10:9

remove 10

create 11

Ok 8693

pingall Ok:10;

exec 11

3 4 5 6

Ok:11:1510:44

# 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 «Pair». 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++.