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: Бонокин Д.С.

Вариант:5

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.

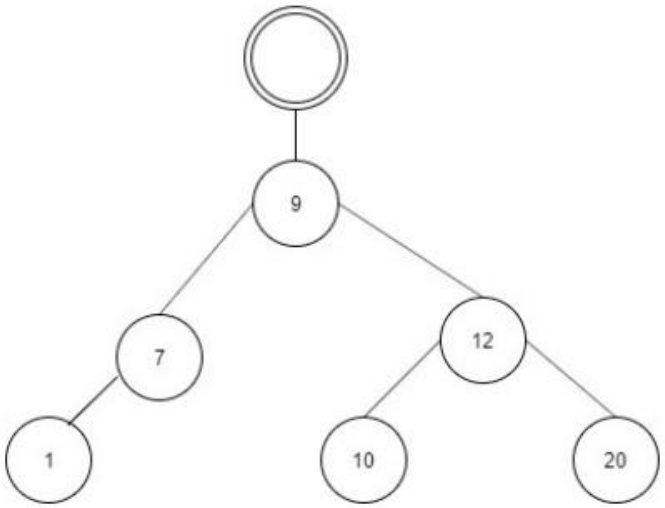
Вариант №5

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

```
danil@danil-HYM-WXX:~/Desktop/lab_os/lab5-6$ ./con
create 10
Ok 8064
create 20
Killed
danil@danil-HYM-WXX:~/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 C++.