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

“Очереди сообщений”

Студент: Зубко Дмитрий Валерьевич
Группа: М8О-208Б-20
Вариант:35
Преподаватель: Миронов Евгений Сергеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2021

Содержание

Репозиторий

https://github.com/usernameMAI/OS/

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.

Вариант 35. Команды:
create id
exec id n n1 n2… ni (набор чисел, требуется посчитать сумму)
ping id

Общие сведения о программе

Программа содержит 6 файлов с кодом:

ServerProgram.cpp – программа сервера

ClientProgram.cpp – программа клиента

CalculationNode.h – программа, в которой реализованы основные команды

ZMQFunctions.h – файл с функциями zero MQ

BalancedTree.h – программа с идеально сбалансированным деревом

Общий метод и алгоритм решения

Программа содержит makefile:

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

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

Создаётся две программы. Запускается client, который вызывает server от определенных значений client_id, parent_port и parent_id. Когда клиент получает сообщение, то он отправляет его на сервер. Все операции проходят с объектом node, который принадлежит классу CalculationNode.

Исходный код

ClientProgram.cpp

```
#include <bits/stdc++.h>
#include "CalculationNode.h"
#include "ZMQFunctions.h"
#include "BalancedTree.h"

using namespace std;

/*

    ВАРИАНТ 35
    4 1 2
    все вычислительные узлы хранятся в идеально сбалансированном бинарном
дереве,
    каждый следующий узел должен добавляться в самое наименьшее поддерево.

    тип команды -- подсчет суммы n чисел
    exec id n k1 k2 ... kn
    id -- целочисленный идентификатор вычислительного узла, на который
отправляется команда
    n -- количество складываемых чисел
    k1 ... kn -- складываемые числа

    формат команды ping id
```

```
    команда проверяет доступность конкретного узла, если узла нет, то
необходимо
    выводить ошибку.

*/

int main() {

    string command, ans;
    BalancedTree tree;
    CalculationNode node(-1, -1, -1);

    cout << "Enter the command: \n\n";
    cout << "create id: for creating a new calculation node\n";
    cout << "exec id n n1 n2... n: for calculating a sum\n";
    cout << "ping id: for checking node-availabilty\n";
    cout << "kill id: for killing a calculation node\n\n";

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

        if (command == "create") {
            int child;
            cin >> child;
            if (tree.Exist(child)) {
                cout << "Error: Already exists" << std::endl;
            } else {
                while (true) {
                    int idParent = tree.FindID();
                    if (idParent == node.id) {
                        ans = node.create(child);
                        tree.AddInTree(child, idParent);
                        break;
                    } else {
                        string message = "create " + to_string(child);
                        ans = node.sendstring(message, idParent);
                        if (ans == "Error: Parent not found") {
                            tree.AvailabilityCheck(idParent);
                        } else {
                            tree.AddInTree(child, idParent);
                            break;
                        }
                    }
                }
                cout << ans << endl;
            }
        } else if (command == "exec") {
            string str;
            int child;
            cin >> child;
            getline(cin, str);
            if (!tree.Exist(child)) {
                cout << "Error: Parent is not existed" << std::endl;
            } else {
                string message = "exec " + str;
                ans = node.sendstring(message, child);
                cout << ans << endl;
            }
        } else if (command == "ping") {
            int child;
            cin >> child;
            if (!tree.Exist(child)) {
                cout << "Error: Parent is not existed" << endl;
            } else if (node.left_id == child || node.right_id == child) {
```

```
                ans = node.ping(child);
                cout << ans << endl;
            } else {
                string message = "ping " + to_string(child);
                ans = node.sendstring(message, child);
                if (ans == "Error: Parent not found") {
                    ans = "Ok: 0";
                }
                cout << ans << endl;
            }
        } else if (command == "kill") {
            int child;
            std::cin >> child;
            std::string message = "kill";
            if (!tree.Exist(child)) {
                std::cout << "Error: Parent is not existed" << std::endl;
            } else {
                ans = node.sendstring(message, child);
                if (ans != "Error: Parent not found") {
                    tree.RemoveFromRoot(child);
                    if (child == node.left_id) {
                        unbind(node.left, node.left_port);
                        node.left_id = -2;
                        ans = "Ok";
                    } else if (child == node.right_id) {
                        node.right_id = -2;
                        unbind(node.right, node.right_port);
                        ans = "Ok";
                    } else {
                        message = "clear " + std::to_string(child);
                        ans = node.sendstring(message, std::stoi(ans));
                    }
                    std::cout << ans << std::endl;
                }
            }
        } else {
            std::cout << "Please enter correct command!" << std::endl;
        }

    }

    node.kill();
}
```

BalancedTree.h

```
#ifndef BALANCED_TREE_H
#define BALANCED_TREE_H

#include <bits/stdc++.h>

using namespace std;

/*

    дерево идеально сбалансировано, если для каждого его уровня число узлов
    в левом и правом поддеревьях отличается не более чем на 1.

*/

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(std::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, (cur-
rent_height + 1));
                }
                if (right != NULL && right->available == true && current_id
== -2) {
                    current_id = right->IDMinimalHeight(height, (cur-
rent_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"

using namespace std;

class CalculationNode {
private:
```

```
//  управляют сокетами
    zmq::context_t context;
public:
//  для связи между процессами

    zmq::socket_t left, right, parent;
    int id, left_id = -2, right_id = -2, parent_id;
    int left_port, right_port, parent_port;

    CalculationNode(int id, int parent_port, int parent_id) :
            id(id),
            parent_port(parent_port),
            parent_id(parent_id),

//          идут парами
//          1 для связи с вверхом, 2 и 3 для связи с потомками

            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("./server", "server", std::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" << std::endl;
                exit(EXIT_FAILURE);
            }
        } else {
            std::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) {
                std::string fail = "Error: can not connect to the child";
                return fail;
            }
        }
    }

    std::string ping(int id) {
        std::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) {
            std::string message = "ping " + std::to_string(id);
            send_message(right, message);
            try {
                message = receive_message(right);
                if (message == "Ok: 1") {
                    answer = message;
                }
            }
            catch (int) {}
        }
        return answer;
    }

    std::string sendstring(std::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 " + std::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 " + std::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;
    }

    std::string exec(std::string string) {
        std::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: " + std::to_string(id) + ": " +
std::to_string(result);
        return answer;
    }

    std::string treeclear(int child) {
        if (left_id == child) {
            left_id = -2;
            unbind(left, left_port);
        } else {
            right_id = -2;
            unbind(right, right_port);
        }
        return "Ok";
    }

    std::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 std::to_string(parent_id);
    }

    ~CalculationNode() {}
};
```

ServerProgram.cpp

```
#include <bits/stdc++.h>
#include "CalculationNode.h"
#include "ZMQFunctions.h"
#include "BalancedTree.h"

using namespace std;

int main(int argc, char *argv[]) {
    if (argc != 4) {
        std:: cout << "Usage: 1)./main, 2) child_id, 3) parent_port, 4) par-
ent_id" << std:: endl;
        exit(EXIT_FAILURE);
    }
    CalculationNode node(atoi(argv[1]), atoi(argv[2]), atoi(argv[3]));
    while(true) {
        std:: string message;
        std:: string command;
        message = receive_message(node.parent);
        std:: istringstream request(message);
        request >> command;
        if (command == "pid") {
            std:: string answer = std:: to_string(getpid());
            send_message(node.parent, answer);
        }
        else if (command == "ping") {
            int child;
            request >> child;
            std:: string answer = node.ping(child);
            send_message(node.parent, answer);
        }
        else if (command == "create") {
            int child;
            request >> child;
            std:: string answer = node.create(child);
            send_message(node.parent, answer);
        }
        else if (command == "send"){
            int child;
            std:: string str;
```

```
            request >> child;
            getline(request, str);
            str.erase(0, 1);
            std:: string answer = node.sendstring(str, child);
            send_message(node.parent, answer);
        }
        else if (command == "exec") {
            std:: string str;
            getline(request, str);
            std:: string answer = node.exec(str);
            send_message(node.parent, answer);
        }
        else if (command == "kill") {
            std:: string answer = node.kill();
            send_message(node.parent, answer);
            disconnect(node.parent, node.parent_port);
            node.parent.close();
            break;
        }
        else if (command == "clear") {
            int child;
            request >> child;
            std:: string answer = node.treeclear(child);
            send_message(node.parent, answer);
        }
    }
    return 0;
}
```

ZMQFunctions.h

```
#pragma once
#include <bits/stdc++.h>
#include <zmq.hpp>
const int MAIN_PORT = 4040;

void send_message(zmq::socket_t &socket, const std::string &msg) {
    zmq::message_t message(msg.size());
    memcpy(message.data(), msg.c_str(), msg.size());
    socket.send(message);
}

std::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;
    }
    std::string received_msg(static_cast<char*>(message.data()), mes-
sage.size());
    return received_msg;
}

void connect(zmq::socket_t &socket, int port) {
```

```
    std::string address = "tcp://127.0.0.1:" + std::to_string(port);
    socket.connect(address);
}

void disconnect(zmq::socket_t &socket, int port) {
    std::string address = "tcp://127.0.0.1:" + std::to_string(port);
    socket.disconnect(address);
}

int bind(zmq::socket_t &socket, int id) {
    int port = MAIN_PORT + id;
    std::string address = "tcp://127.0.0.1:" + std::to_string(port);
    while(1){
        try{
            socket.bind(address);
            break;
        }
        catch(...){
            port++;
        }
    }
    return port;
}

void unbind(zmq::socket_t &socket, int port) {
    std::string address = "tcp://127.0.0.1:" + std::to_string(port);
    socket.unbind(address);
}
```

Демонстрация работы программы

Enter the command:

create id: for creating a new calculation node

exec id n n1 n2... n: for calculating a sum

ping id: for checking node-availabilty

kill id: for killing a calculation node

Enter command: create 10

Ok: 61

Enter command: create 15

Ok: 64

Enter command: create 11

Ok: 67

Enter command: create 12

Ok: 70

Enter command: create 13

Ok: 73

Enter command: create 14

Ok: 76

Enter command: exec 14 5 1 2 3 4 5

Ok: 14: 15

Enter command: ping 15

Ok: 1

Enter command:

Выводы

Благодаря данной лабораторной работе я узнал, что такое синхронные и асинхронные вычисления. Приобрел практические навыки работы с zero message queue. Закрепил навыки, полученные в предыдущих лабораторных работах.