Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №5-7 по курсу**

**«Операционные системы»**

**УПРАВЛЕНИЕ СЕРВЕРАМИ СООБЩЕНИЙ. ПРИМЕНЕНИЕ ОТЛОЖЕННЫХ ВЫЧИСЛЕНИЙ. ИНТЕГРАЦИЯ ПРОГРАММНЫХ СИСТЕМ ДРУГ С ДРУГОМ.**

Студент: Козырев Пётр Андреевич

Группа: М8О–212Б–22

Вариант: 47

Преподаватель: Соколов Андрей Алексеевич

Оценка: \_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_

Москва, 2023.

**Постановка задачи**

## Цель работы

Целью является приобретение практических навыков в:

* Управлении серверами сообщений (№5)
* Применение отложенных вычислений (№6)
* Интеграция программных систем друг с другом (№7)

## Задание

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.

Набора команд 3 (локальный таймер):

Формат команды сохранения значения:

exec id subcommand

subcommand – одна из трех команд: start, stop, time.

start – запустить таймер

stop – остановить таймер

time – показать время локального таймера в миллисекундах

Тип проверки доступности узлов

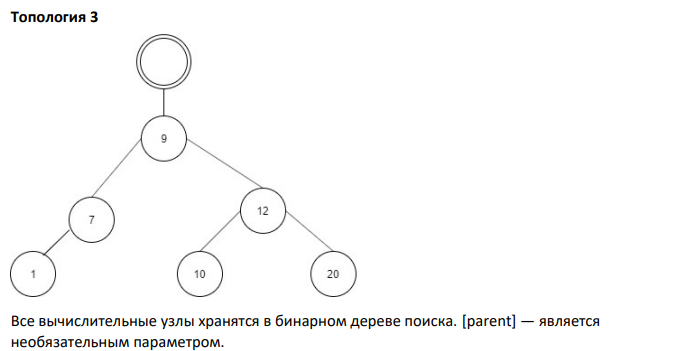
Команда проверки 1

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



**Общие сведения о программе**

*Tree\_elem.h,binary\_tree.h, timer.h –* заголовочные файлы;

*Binary\_tree.cpp –* реализация бинарного дерева;

*Timer.cpp –* реализация таймера;

*Client.cpp –* вычислительный узел;

*Main.cpp –* управляющий узел;

*Makefile –* сборка программы.

**Общий метод и алгоритм решения**.

1. Изучить библиотеку ZeroMQ для работы с очередями сообщений
2. Реализовать таймер
3. Реализовать бинарное дерево в соответствии с топологией задания
4. Реализовать главный управляющий узел, который принимает команды со стандартного входа и отправляет их на вычислительные узлы.
5. Реализовать вычислительный узел, который принимает команды от управляющего узла и выполняет их или перенаправляет команды на дочерние узлы.

**Основные файлы программы**

**Main.cpp**

#include <iostream>

#include <cerrno>

#include <cstring>

#include <string>

#include <vector>

#include <algorithm>

#include <sstream>

#include <assert.h>

#include <signal.h>

#include <string.h>

#include <unistd.h>

#include "zmq.h"

#include "../input/binary\_tree.h"

typedef struct{

    int command;

    int subcommand;

    int node;

} data\_for\_node;

int create\_node(int id, int parent)

{

    std::string id\_s = std::to\_string(id);

    std::string parent\_s = std::to\_string(parent);

    const char\* arg0 = "./client";

    const char\* arg1 = id\_s.c\_str();

    const char\* arg2 = parent\_s.c\_str();

    if(execl("./client", arg0, arg1, arg2, NULL) == -1){

        return -1;

    }

    return 1;

}

int main(){

    binary\_tree bin\_tree(0);

    std::string command;

    int id;

    pid\_t child\_pid;

    int child\_id = 0;

    int parent\_id = 0;

    std::vector<int> buzy\_node;

    void\* context = zmq\_ctx\_new();

    void\* publiser = zmq\_socket(context, ZMQ\_PUB);

    int rc = zmq\_bind(publiser, "tcp://\*:5000");

    if(rc == -1){

        std::cerr << std::strerror(errno) << std::endl;

    }

    assert(rc == 0);

    std::cout << "=== Commands === \n";

    std::cout << "create id\n";

    std::cout << "exec id subcommand (start/stop/time)\n";

    std::cout << "kill id\n";

    std::cout << "pingall\n";

    std::cout << "exit\n" << std::endl;

    while(true){

        std::cin >> command;

        if(command == "create"){

            std::cin >> id;

            std::cout << std::endl;

            if(bin\_tree.find(id)){

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

                continue;

            }

            bin\_tree.insert(id);

            parent\_id = bin\_tree.parent(id);

            if(parent\_id == -1){

                std::cout << "Error: Parent not found" << std::endl;

                continue;

            }

            child\_pid = fork();

            if(child\_pid > 0){

                std::cout << "Ok: " << child\_pid << std::endl;

            }

            else if(child\_pid == 0){

                if(create\_node(id, parent\_id) == -1){

                    std::cout << "Error: in executing new node" << std::endl;

                    continue;

                }

            }

            else{

                std::cout << "Error: fork error" << std::endl;

                continue;

            }

        }

        else if(command == "exec"){

            std::string subcommand;

            std::cin >> id >> subcommand;

            std::cout << std::endl;

            if(!bin\_tree.find(id)){

                std::cout << "«Error:id: Not found" << std::endl;

                continue;

            }

            data\_for\_node\* data = (data\_for\_node\*)malloc(sizeof(data\_for\_node));

            data->command = 1;

            if(subcommand == "start"){

                data->subcommand = 1;

                buzy\_node.push\_back(id);

            }

            else if(subcommand == "stop"){

                data->subcommand = 2;

                buzy\_node.erase(std::remove(buzy\_node.begin(), buzy\_node.end(), id), buzy\_node.end());

            }

            else if(subcommand == "time") data->subcommand = 3;

            else{

                std::cout << "Error: Wrong subcommand" << std::endl;

                continue;

            }

            data->node = id;

            zmq\_msg\_t message;

            zmq\_msg\_init\_size(&message, sizeof(data\_for\_node));

            memcpy(zmq\_msg\_data(&message), data, zmq\_msg\_size(&message));

            zmq\_msg\_send(&message, publiser, 0);

            zmq\_msg\_close(&message);

            free(data);

        }

        else if(command == "pingall"){

            if(buzy\_node.size() == 0){

                std::cout << "Ok: -1" << std::endl;

            }

            else{

                std::cout << "Ok: ";

                for(int elem: buzy\_node){

                    std::cout << elem << "; ";

                }

                std::cout << std::endl;

            }

        }

        else if(command == "kill"){

            std::cin >> id;

            std::cout << std::endl;

            if(!bin\_tree.find(id)){

                std::cout << "Error: doesn`t exist" << std::endl;

                continue;

            }

            buzy\_node.erase(std::remove(buzy\_node.begin(), buzy\_node.end(), id), buzy\_node.end());

            bin\_tree.erase(id);

            data\_for\_node\* data = (data\_for\_node\*)malloc(sizeof(data\_for\_node));

            data->command = 2;

            data->node = id;

            zmq\_msg\_t message;

            zmq\_msg\_init\_size(&message, sizeof(data\_for\_node));

            memcpy(zmq\_msg\_data(&message), data, zmq\_msg\_size(&message));

            zmq\_msg\_send(&message, publiser, 0);

            zmq\_msg\_close(&message);

            free(data);

        }

        else if(command == "exit"){

            int n = system("killall client");

            break;

        }

    }

    zmq\_close(publiser);

    zmq\_ctx\_destroy(context);

    return 0;

}

**Client.cpp**

#include <iostream>

#include <unistd.h>

#include <string>

#include <string.h>

#include <sstream>

#include <stdlib.h>

#include <exception>

#include <assert.h>

#include <stdlib.h>

#include <signal.h>

#include <cerrno>

#include <cstring>

#include "zmq.h"

#include "../input/timer.h"

int PORT = 5000;

typedef struct{

    int command;

    int subcommand;

    int node;

} data\_for\_node;

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

    if(argc != 3){

        std::cerr << "Error: few arguments" << std::endl;

        return 1;

    }

    int CURRENT\_NODE = atoi(argv[1]);

    int PARENT\_NODE = atoi(argv[2]);

    int rc;

    Timer timer;

    void\* context = zmq\_ctx\_new();

    void\* subscriber = zmq\_socket(context, ZMQ\_SUB);

    std::string adress\_par = "tcp://localhost:" + std::to\_string(PORT + PARENT\_NODE);

    rc = zmq\_connect(subscriber, adress\_par.c\_str());

    assert(rc == 0);

    rc = zmq\_setsockopt(subscriber, ZMQ\_SUBSCRIBE,"",0);

    assert(rc == 0);

    void\* publisher = zmq\_socket(context, ZMQ\_PUB);

    std::string adress\_cur = "tcp://\*:" + std::to\_string(PORT + CURRENT\_NODE);

    rc = zmq\_bind(publisher, adress\_cur.c\_str());

    if(rc == -1){

        std::cerr << std::strerror(errno) << std::endl;

    }

    assert(rc == 0);

    for(;;){

        zmq\_msg\_t reply;

        zmq\_msg\_init(&reply);

        zmq\_msg\_recv (&reply, subscriber, 0);

        data\_for\_node\* data = (data\_for\_node\*)malloc(zmq\_msg\_size(&reply));

        memcpy(data, zmq\_msg\_data(&reply), zmq\_msg\_size(&reply));

        if (data->command == 1){

            if(data->node != CURRENT\_NODE){

                zmq\_msg\_send(&reply, publisher, 0);

                zmq\_msg\_close(&reply);

                free(data);

                continue;

            }

            zmq\_msg\_close(&reply);

            if(data->subcommand == 1){

                timer.start\_timer();

                std::cout << "Ok: " << CURRENT\_NODE << std::endl;

            }

            else if(data->subcommand == 2){

                timer.stop\_timer();

                std::cout <<"Ok: " << CURRENT\_NODE << std::endl;

            }

            else{

                std::cout << "Ok: " << CURRENT\_NODE << ": " << timer.get\_time() << "ms" << std::endl;

            }

        }

        else if(data->command == 2){

            if(data->node == CURRENT\_NODE || data->node == -1){

                printf("Ok:%d\n", CURRENT\_NODE);

                zmq\_msg\_close(&reply);

                free(data);

                data\_for\_node\* data2 = (data\_for\_node\*)malloc(sizeof(data\_for\_node));

                data2->command = 2;

                data2->node = -1;

                zmq\_msg\_t message;

                zmq\_msg\_init\_size(&message, sizeof(data\_for\_node));

                memcpy(zmq\_msg\_data(&message), data2, zmq\_msg\_size(&message));

                zmq\_msg\_send(&message, publisher, 0);

                zmq\_msg\_close(&message);

                free(data);

                break;

            }

            else {

                zmq\_msg\_send(&reply, publisher, 0);

                zmq\_msg\_close(&reply);

            }

        }

        else if(data->command == 2){

            if(data->node == CURRENT\_NODE){

                zmq\_close(publisher);

            }

        }

        free(data);

    }

    zmq\_close(subscriber);

    zmq\_close(publisher);

    zmq\_ctx\_destroy(context);

    return 0;

}

**Пример работы**

**petrkozyrev@LAPTOP-F4OC0VPH:~/osi/os\_lab\_5-7/src$ ./main**

**=== Commands ===**

**create id**

**exec id subcommand (start/stop/time)**

**kill id**

**pingall**

**exit**

**create 5**

**Ok: 3329**

**create 10**

**Ok: 3340**

**pingall**

**Ok: -1**

**exec 5 start**

**Ok: 5**

**pingall**

**Ok: 5;**

**exec 5 time**

**Ok: 5: 8985ms**

**exec 5 stop**

**Ok: 5**

**pingall**

**Ok: -1**

**kill 10**

**Ok:10**

**exec 5 start**

**Ok: 5**

**pingall**

**Ok: 5;**

**exec 5 stop**

**Ok: 5**

**exit**

**Вывод**

В результате выполнения лабораторной работы мной были приобретены практические навыки в управлении серверами сообщений, применении отложенных вычислений и интеграции программных систем друг с другом.

Я познакомился с технологией очередей сообщений и изучил библиотеку ZeroMQ.

В результате лабораторной работы мной была реализована распределенная система по асинхронной обработке запросов в соответствии с заданием.