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»**

Студент: Путилин Д.Н.
Группа: М8О-207Б-21
Вариант: 36
Преподаватель: Миронов Евгений Сергеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2022

**Содержание**

## Репозиторий

https://github.com/putilin21dn/OC

## Постановка задачи

## Цель работы

Целью является приобретение практических навыков в:

- Управлении серверами сообщений (№6)
- Применение отложенных вычислений (№7)
- Интеграция программных систем друг с другом (№8)

## 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. Список основных поддерживаемых команд: create, exec, remove, heartbit.

## Общие сведения о программе

Программа распределительного узла компилируется из файла annotations_node.cpp, программа вычислительного узла компилируется из файла compiling_node.cpp. В программе используется библиотека для работы с потоками и мьютексами, а также сторонняя библиотека для работы с сервером сообщений ZeroMQ.

## Общий метод и алгоритм решения

Распределительный узел хранит в себе двоичное дерево, элементами которого являются async_node – структуры, хранящие в себе id вычислительного узла, порт для связи с ним, поток отправляющий/принимающий запросы/ответы к узлу, очередь отправленных на узел запросов и мьютекс, обеспечивающий возможность работать с этой очередью в несколько потоков.

В функции main запускается бесконечный цикл обработки пользовательских запросов, при получении запроса он отправляется в очередь обработчика соответствующего узла, откуда позже будет извлечён потоком обработки и переслан требуемому узлу (обеспечение асинхронности). При удалении узла у записи, соответствующей удаляемому узлу устанавливается переменная активности в false, благодаря чему обработчик, отправив на узел запрос удаления и получив ответ, выходит из цикла и очищает память от уже ненужной записи. Подобные действия применяются и ко всем дочерним узлам удаляемого.

Heartbit. После запуска команды и введенного времени создается поток, который будет через время time проходится по всем узлам и проверяет их на установку соединения. В результате у нас

в случае усспеха узел будет иметь 1, в противном случае – 0. Дальше уже в следующих командах мы можем удалять или добавлять узлы. Heartbit будет работать также  и с ними. Для проверки работоспособности узла используем команду ping, которая в случае успеха возвращет Ок:1, иначе Ок:-1.

## Исходный код

**annotations_node.cpp**

```
#include "zmq.h"
#include "string.h"
#include "unistd.h"
#include "stdlib.h"
#include "pthread.h"

#include <iostream>
#include <queue>
#include <vector>
#include "header.hpp"
#include <ctime>
#include <cstdlib>

#define CHECK_ERROR(expr, stream, act) \
    do \
    { \
        int res = (expr); \
        if (res != 0) \
        { \
            std::cerr << stream; \
            act; \
        } \
    } while (0)

#define CHECK_ZMQ(expr, stream, act) \
    do \
    { \
        int res = (expr); \
        if (res == -1) \
        { \
            std::cerr << stream; \
            act; \
        } \
    } while (0)



using namespace std;

string protocol = "tcp://localhost:";
int MIN_PORT;
vector<pair<int,bool>> act;
int times;
bool flag = true;
void* async_node_thd(void*);



struct async_node
{
    int id;
    string port;
    bool active;
    async_node* L;
    async_node* R;
    pthread_mutex_t mutex;
    pthread_t thd;
```

```
    queue <vec> q;




    async_node(int i)
    {
        id = i;
        port = protocol + to_string(i);
        active = true;
        L = nullptr;
        R = nullptr;
        CHECK_ERROR(pthread_mutex_init(&mutex, NULL), "Error:" << i - MIN_PORT << ": Gateway mutex
error\n", return);
        CHECK_ERROR(pthread_create(&thd, NULL, async_node_thd, this), "Error:" << i - MIN_PORT <<
": Gateway thread error\n", return);
        CHECK_ERROR(pthread_detach(thd), "Error:" << i << ": Gateway thread error\n", return);
    }

    void make_query(vec V)
    {
        CHECK_ERROR(pthread_mutex_lock(&mutex), "Error:" << id - MIN_PORT << ": Gateway mutex lock
error\n", active = false; return);
        q.push(V);
        CHECK_ERROR(pthread_mutex_unlock(&mutex), "Error:" << id - MIN_PORT << ": Gateway mutex
unlock error\n", active = false);
    }

    ~async_node()
    {
        pthread_mutex_destroy(&mutex);
    }
};

async_node* find_node_exec(async_node* ptr, int id)
{
    if (ptr == nullptr)
        return nullptr;
    if (ptr->id > id)
        return find_node_exec(ptr->L, id);
    if (ptr->id < id)
        return find_node_exec(ptr->R, id);
    return ptr;
}

async_node* find_node_create(async_node* ptr, int id)
{
    if (ptr == nullptr)
        return nullptr;
    if (ptr->L == nullptr && ptr->id > id)
        return ptr;
    if (ptr->R == nullptr && ptr->id < id)
        return ptr;
    if (ptr->id > id)
        return find_node_create(ptr->L, id);
    if (ptr->id < id)
        return find_node_create(ptr->R, id);
    return nullptr;
}

bool destroy_node(async_node*& ptr, int id)
{
    if (ptr == nullptr)
        return false;
    if (ptr->id > id)
        return destroy_node(ptr->L, id);
    if (ptr->id < id)
        return destroy_node(ptr->R, id);
```

```
    ptr->active = false;
    ptr->make_query({REMOVE});
    if (ptr->L != nullptr)
        destroy_node(ptr->L, ptr->L->id);
    if (ptr->R != nullptr)
        destroy_node(ptr->R, ptr->R->id);
    ptr = nullptr;
    return true;
}
bool pings(int);

void* async_node_thd(void* ptr)
{
    async_node* node = (async_node*)ptr;
    void* context = zmq_ctx_new();
    void* req = zmq_socket(context, ZMQ_REQ);
    CHECK_ZMQ(zmq_connect(req, node->port.c_str()), "Error: Connection with" << node->id -
MIN_PORT << "\n",);
    while (node->active)
    {
        if (node->q.empty())
            continue;
        CHECK_ERROR(pthread_mutex_lock(&node->mutex), "Error:" << node->id - MIN_PORT << ": Gate-
way mutex lock error\n", node->active = false; break);
        vec V = node->q.front();
        node->q.pop();
        CHECK_ERROR(pthread_mutex_unlock(&node->mutex), "Error:" << node->id - MIN_PORT << ":
Gateway mutex unlock error\n", node->active = false; break);

        switch (V.ex)
        {
            case CREATE:
            {
                zmq_msg_t msg;
                msg = vec2msg(V);
                CHECK_ZMQ(zmq_msg_send(&msg, req, 0), "Error:" << node->id - MIN_PORT << ": Mes-
sage error\n", break);
                int pid;
                CHECK_ZMQ(zmq_recv(req, &pid, sizeof(int), 0), "Error:" << node->id - MIN_PORT <<
": Message error\n", break);
                if (V.id < node->id)
                    node->L = new async_node(V.id);
                else
                    node->R = new async_node(V.id);
                cout << "Ok: " << pid << '\n';
                zmq_msg_close(&msg);
                break;
            }

            case EXEC:
            {
                zmq_msg_t msg;
                msg = vec2msg(V);
                CHECK_ZMQ(zmq_msg_send(&msg, req, 0), "Error:" << node->id - MIN_PORT << ": Mes-
sage error\n", break);
                string ans;
                CHECK_ZMQ(zmq_msg_recv(&msg, req, 0), "Error:" << node->id - MIN_PORT << ": Mes-
sage error\n", break);
                cout << "Ok:" << node->id - MIN_PORT << ":";
                ans = msg2str(msg);

                for(int i=0; i<ans.length();++i){
                    if(ans[i]!='#')
                        cout << ans[i];
                    else{
                        if(i==(ans.length()-1))
                            cout << '\n';
                        else{
```

```
                            cout << ";";
                        }
                    }
                }

                zmq_msg_close(&msg);
                break;
            }

            case REMOVE:
            {
                CHECK_ZMQ(zmq_send(req, &V.ex, sizeof(int), 0), "Error:" << node->id - MIN_PORT <<
": Message error\n", break);
                int ans;
                CHECK_ZMQ(zmq_recv(req, &ans, sizeof(int), 0), "Error:" << node->id - MIN_PORT <<
": Message error\n", break);
                break;
            }
        }
    }
    zmq_close(req);
    zmq_ctx_destroy(context);
    delete node;
    return NULL;
}
async_node* tree = nullptr;

struct th{
    async_node *tree;
    int time;
};



bool pings(int id)
{


        string port = protocol + to_string(id);
        string ping = "inproc://ping" + to_string(id);
        void* context = zmq_ctx_new();
        void *req = zmq_socket(context, ZMQ_REQ);

        zmq_socket_monitor(req, ping.c_str(), ZMQ_EVENT_CONNECTED | ZMQ_EVENT_CONNECT_RETRIED);
        void *soc = zmq_socket(context, ZMQ_PAIR);
        zmq_connect(soc, ping.c_str());
        zmq_connect(req, port.c_str());

        zmq_msg_t msg;
        zmq_msg_init(&msg);
        zmq_msg_recv(&msg, soc, 0);
        uint8_t* data = (uint8_t*)zmq_msg_data(&msg);
        uint16_t event = *(uint16_t*)(data);

        zmq_close(req);
        zmq_close(soc);
        zmq_msg_close(&msg);
        zmq_ctx_destroy(context);

        return event % 2;
}


void ping(int id)
    {
        for (auto x : act){
            if(x.first == id){
```

```
                cout << "Ok : 1" << '\n';
                return;
            }
        }
        cout << "Ok : -1" << '\n';
        return;
    }



void * heartbit(void *args){
    // int times = *((int*)(&args));
    int time = ((th *)args)->time;
    async_node* node = ((th *)args)->tree;
    while(times>0){
        queue <async_node*> q;
        if (node != nullptr)
            q.push(node);
            act.resize(0);

            while (!q.empty())
        {
            async_node* ptr = q.front();
            q.pop();
            if (ptr->L != nullptr)
                q.push(ptr->L);
            if (ptr->R != nullptr)
                q.push(ptr->R);
            bool check = pings(ptr->id);

            act.push_back({ptr->id,check});
        }
        sleep(4*times/1000);
    }
    return NULL;
}



int main()
{
    srand( time(0) );
    MIN_PORT = 1024 + rand()%1000;

    while (true)
    {
        string command;
        cin >> command;


        if (command == "create")
        {
            int id;
            cin >> id;
            id += MIN_PORT;
            vec V;
            V.ex = CREATE;
            V.id = id;
            if (tree == nullptr)
            {
                string id_str = to_string(id);
                int pid = fork();
                if (pid == 0)
                    CHECK_ERROR(execl("server", id_str.c_str(), NULL), "Error:" << id - MIN_PORT
<< ": Creating error\n", break);
                cout << "Ok: " << pid << '\n';
                tree = new async_node(id);
            }
```

```
            else
            {
                async_node* node = find_node_create(tree, id);
                if (node != nullptr){
                    if (!pings(node->id))
                    {
                        cerr << "Error:" << id - MIN_PORT << ": Parent is unavailable\n";
                        continue;
                    }
                    node->make_query({CREATE, id});
                }
                else
                    cerr << "Error: Already exists\n";
            }
        }

        if (command == "exec")
        {
            int id;
            cin >> id;
            id += MIN_PORT;
            string str,substr;
            cin >> str >> substr;
            vec V;
            V.ex = EXEC;
            V.id = id;
            V.lenstr = str.length();
            V.str = str;
            V.lensubstr = substr.length();
            V.substr = substr;
            if (!pings(id))
            {
                cerr << "Error:" << id - MIN_PORT << ": Node is unavailable\n";
                continue;
            }
            async_node* node = find_node_exec(tree, id);
            if (node != nullptr)
                node->make_query(V);
            else
                cerr << "Error:" << id - MIN_PORT << ": Not found\n";
        }

        if (command == "remove")
        {
            int id;
            cin >> id;
            id += MIN_PORT;
            if (!pings(id))
            {
                cerr << "Error:" << id - MIN_PORT << ": Node is unavailable\n";
                continue;
            }
            bool state = destroy_node(tree, id);
            if (state)
                cout << "Ok\n";
            else
                cerr << "Error: Not found\n";
        }


        if (command == "heartbit"){

            cin >> times;
            vector<pthread_t> threads = vector<pthread_t>(1);
            th V;
            V.time = times;
            V.tree = tree;
            if(flag){
```

```
                if (times>0){

                    flag = false;
                    if(int err = pthread_create(&threads[0],NULL, heartbit, (void *)&V))

                    if (pthread_join(threads[0],NULL) != 0) {
                        cout << "Can't wait for thread\n";
                    }
                }
            }
        }

        if(command == "ping"){
            int id;
            cin >> id;
            id += MIN_PORT;
            ping(id);
        }
    }
}
```

**compiling_node.cpp**

```
#include <zmq.h>
#include <unistd.h>
#include <string.h>
#include "header.hpp"
#include <iostream>
#include <vector>

#define CREATE 1
#define EXEC 0
#define REMOVE -1

#define CHECK_ERROR(expr) \
    do \
    { \
        int res = (expr); \
        if (res == -1){ \
            std::cout << errno << '\n'; \
            return -1;} \
    } while (0)

using namespace std;

string z_func(int lensubstr, string &s){
    int n = s.size();
    vector<int> z(n,0);
    string ans;
    int l=-1,r=-1;
    z[0]=n;
    for(int i=1;i<n;++i){
        if(i<=r){
            z[i]=min(z[i-l], r-i);
        }
        while((i+z[i]<n) && (s[z[i]]==s[i+z[i]])){
            ++z[i];
        }
        if(i+z[i]>r){
            r=i+z[i];
            l=i;
        }
    }
    for(int i=lensubstr; i<n;++i){
        if(z[i]==lensubstr){
```

```
            ans += to_string(i-lensubstr-1) + "#";
        }
    }
    return ans;
}


int main(int argc, char* argv[])
{
    int id = atoi(argv[0]);
    string port = "tcp://*:" + to_string(id);
    void *context = zmq_ctx_new();
    void *responder = zmq_socket(context, ZMQ_REP);
    zmq_bind(responder, port.c_str());

    while (true)
    {
        zmq_msg_t msg;
        CHECK_ERROR(zmq_msg_init(&msg));
        // cout << "heresa" <<'\n';
        CHECK_ERROR(zmq_msg_recv(&msg, responder, 0));
        // cout << "heresdfs" <<'\n';
        vec V = msg2vec(msg);




        switch (V.ex)
        {
            case CREATE:
            {
                string id_str = to_string(V.id);
                // cout << "here2" << '\n';
                int pid = fork();
                // cout << "here 2" << pid <<'\n';
                if (pid == -1)
                    return -1;
                if (pid == 0)
                    CHECK_ERROR(execl("server", id_str.c_str(), NULL));
                else
                    CHECK_ERROR(zmq_send(responder, &pid, sizeof(int), 0));
                break;
            }

            case EXEC:
            {
                string s = V.substr + "#" + V.str;
                string ans;
                ans = z_func(V.lensubstr,s);

                zmq_msg_t msg = str2msg(ans);
                zmq_msg_send(&msg,responder,0);
                break;
            }

            case REMOVE:
            {
                zmq_send(responder, &id, sizeof(int), 0);
                zmq_close(responder);
                zmq_ctx_destroy(context);
                return 0;
            }
        }
        CHECK_ERROR(zmq_msg_close(&msg));
    }
}
```

**Header.hpp**

```
#ifndef CONSTANTS_HPP
#define CONSTANTS_HPP

#include <string>
#include <zmq.h>
#include <string.h>
#include <iostream>
using namespace std;

#define CREATE 1
#define EXEC 0
#define REMOVE -1

struct vec{
   int ex;
   int id;
   int lenstr;
   string str;
   int lensubstr;
   string substr;
};

zmq_msg_t vec2msg(vec V){

   zmq_msg_t msg;
   zmq_msg_init_size(&msg, 4 * sizeof(int) + V.lenstr+V.lensubstr);
   void * z = zmq_msg_data(&msg);
   memcpy(z, &V.ex, sizeof(int));
   memcpy(z = (void *)((unsigned long)z + sizeof(int)), &V.id, sizeof(int));
   memcpy(z = (void *)((unsigned long)z + sizeof(int)), &V.lenstr, sizeof(int));
   memcpy(z = (void *)((unsigned long)z + sizeof(int)), V.str.c_str(), sizeof(char)*V.lenstr);
   memcpy(z = (void *)((unsigned long)z + sizeof(char)*V.lenstr), &V.lensubstr, sizeof(int));
   memcpy(z = (void *)((unsigned long)z + sizeof(int)), V.substr.c_str(), sizeof(char)*V.lensubstr);
   return msg;

}

vec msg2vec(zmq_msg_t msg){
   void * z = zmq_msg_data(&msg);
   vec V;

   memcpy(&V.ex, z, sizeof(int));
   memcpy(&V.id, z=(void *)((unsigned long)z + sizeof(int)), sizeof(int));
   memcpy(&V.lenstr, z=(void *)((unsigned long)z + sizeof(int)), sizeof(int));
   char *  s = (char *)calloc(V.lenstr,sizeof(char));
   memcpy(s, z = (void *)((unsigned long)z + sizeof(int)) , V.lenstr);
   V.str = s;
   free(s);
   memcpy(&V.lensubstr, z=(void *)((unsigned long)z + sizeof(char)*V.lenstr), sizeof(int));
   s = (char *)calloc(V.lensubstr,sizeof(char));
```

```
    s = s + '\0';
    memcpy(s, z=(void *)((unsigned long)z + sizeof(int)) , V.lensubstr);
    V.substr = s;
    free(s);
    return V;

}

zmq_msg_t str2msg(string str){
    zmq_msg_t msg;
    zmq_msg_init_size(&msg, str.length());
    void * z = zmq_msg_data(&msg);
    memcpy(z, str.c_str(), sizeof(char)*str.length());
    return msg;
}

string msg2str(zmq_msg_t msg){
    void * z = zmq_msg_data(&msg);
    int len = zmq_msg_size(&msg);
    char * s = (char *)calloc(len,sizeof(char));
    memcpy(s,z,len*sizeof(char));
    return s;
}

#endif
```

## Демонстрация работы программы

```
dmitry@dmitry-VirtualBox:~/Рабочий стол/ОС/lab6-8/build$ ./client
create 10
Ok: 9956
create 20
Ok: 9964
create 30
Ok: 9976
exec 30
abasbdbabab
ab
Ok:30:0;7;9
remove 30
Ok
exec 3
asndas
sa
Error:3: Node is unavailable
exec 20
hello
a
```

```
Ok:20:-1
heartbit 1000
ping 20
Ok : 1
ping 20  // after kill 9964
Ok : -1
```

## Выводы

Составлена и отлажена программа на языке C++, осуществляющая отложенные вычисления на нескольких вычислительных узлах. Пользователь управляет программой через распределительный узел, который перенаправляет запросы в асинхронном режиме.