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»

СОЗДАНИЕ РАСПРЕДЕЛЕННОЙ СИСТЕМЫ ПО АСИНХРОННОЙ ОБРАБОТКЕ ЗАПРОСОВ

Студент: Забелкин Андрей Алексеевич
Группа: М8О–210Б–22
Вариант: 37
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.

Согласно моему варианту мои узлы находятся в списке, каждый вычислительный узел должен хранить локальный целочисленный словарь, а проверку работоспособности осуществляться через pingall.

Общие сведения о программе

Программа компилируется из с помощью Makefile, сгенерированным cmake.

В лабораторной работе используются ZMQ.

Общий метод и алгоритм решения.

Для реализации поставленной задачи необходимо:

1. Реализовать вспомогательную библиотеку функций для управление zmq.
2. Реализовать управляющий узел и вычислительный узел.
3. Отладить их работу и взаимодействие.
4. Выработать устойчивость к отключению процессов.

Основные файлы программы

calculation_node.cpp

```
#include <list>
#include <iostream>
#include <pthread.h>
#include <map>
#include <tuple>
#include <unistd.h>

#include "zmq_std.hpp"

const std::string SENTINEL_STR = "$";

long long node_id;

int main(int argc, char** argv) {
	int rc;
	assert(argc == 2);
	node_id = std::stoll(std::string(argv[1]));

	void* node_parent_context = zmq_ctx_new();
	void* node_parent_socket = zmq_socket(node_parent_context, ZMQ_PAIR);
	rc = zmq_connect(node_parent_socket, ("tcp://localhost:" + std::to_string(PORT_BASE +
node_id)).c_str());
	assert(rc == 0);

	long long child_id = -1;
	void* node_context = NULL;
	void* node_socket = NULL;

	std::string value, key;
	bool flag_sentinel = true;

	node_token_t* info_token = new node_token_t({info, getpid(), getpid()});
	zmq_std::send_msg_dontwait(info_token, node_parent_socket);

	std::map<std::string, int> node_map;
	bool has_child = false;
	bool awake = true;
	bool calc = true;
	while (awake) {
		node_token_t token;
		zmq_std::recieve_msg(token, node_parent_socket);

		node_token_t* reply = new node_token_t({fail, node_id, node_id});

		if (token.action == create) {
			if (token.parent_id == node_id) {
				if (has_child) {
					rc = zmq_close(node_socket);
					assert(rc == 0);
					rc = zmq_ctx_term(node_context);
					assert(rc == 0);
				}
				zmq_std::init_pair_socket(node_context, node_socket);
				rc = zmq_bind(node_socket, ("tcp://*:" + std::to_string(PORT_BASE +
token.id)).c_str());
				assert(rc == 0);

				int fork_id = fork();
				if (fork_id == 0) {
					rc = execl(NODE_EXECUTABLE_NAME, NODE_EXECUTABLE_NAME,
std::to_string(token.id).c_str(), NULL);
					assert(rc != -1);
					return 0;
				} else {
					bool ok = true;
					node_token_t reply_info({fail, token.id, token.id});
					ok = zmq_std::recieve_msg_wait(reply_info, node_socket);
					if (reply_info.action != fail) {
```

```
                                        reply->id = reply_info.id;
                                        reply->parent_id = reply_info.parent_id;
                                }
                                if (has_child) {
                                        node_token_t* token_bind = new node_token_t({bind,
token.id, child_id});
                                        node_token_t reply_bind({fail, token.id, token.id});
                                        ok = zmq_std::send_recieve_wait(token_bind, reply_bind,
node_socket);
                                        ok = ok and (reply_bind.action == success);
                                }
                                if (ok) {
                                        /* We should check if child has connected to this node */
                                        node_token_t* token_ping = new node_token_t({ping,
token.id, token.id});
                                        node_token_t reply_ping({fail, token.id, token.id});
                                        ok = zmq_std::send_recieve_wait(token_ping, reply_ping,
node_socket);
                                        ok = ok and (reply_ping.action == success);
                                        if (ok) {
                                                reply->action = success;
                                                child_id = token.id;
                                                has_child = true;
                                        } else {
                                                rc = zmq_close(node_socket);
                                                assert(rc == 0);
                                                rc = zmq_ctx_term(node_context);
                                                assert(rc == 0);
                                        }
                                }
                        }
                } else if (has_child) {
                        node_token_t* token_down = new node_token_t(token);
                        node_token_t reply_down(token);
                        reply_down.action = fail;
                        if (zmq_std::send_recieve_wait(token_down, reply_down, node_socket) and
reply_down.action == success) {
                                *reply = reply_down;
                        }
                }
        } else if (token.action == ping) {
                if (token.id == node_id) {
                        reply->action = success;
                } else if (has_child) {
                        node_token_t* token_down = new node_token_t(token);
                        node_token_t reply_down(token);
                        reply_down.action = fail;
                        if (zmq_std::send_recieve_wait(token_down, reply_down, node_socket) and
reply_down.action == success) {
                                *reply = reply_down;
                        }
                }
        } else if (token.action == exec) {
                if (token.id == node_id) {
                        bool reply_flag = false;
                        char c = token.parent_id;
                        if (c == SENTINEL) {
                                std::cout << "Here";
                                if (flag_sentinel) {
                                        std::swap(key, value);
                                } else {
                                        std::swap(key, value);
                                        if (value == "get") {
                                                auto it = node_map.find(key);
                                                if (it != node_map.end()) {
                                                        reply->parent_id = node_map[key];
                                                } else {
                                                        reply_flag = true;
                                                }
                                        } else {
                                                node_map[key] = std::stoi(value);
```

```
                                        }
                                        key.clear();
                                        value.clear();
                                }
                                flag_sentinel = flag_sentinel ^ 1;
                        } else {
                                key = key + c;
                        }
                        reply->action = success;
                        if (reply_flag) {
                                reply->action = notfound;
                        }
                } else if (has_child) {
                        node_token_t* token_down = new node_token_t(token);
                        node_token_t reply_down(token);
                        reply_down.action = fail;
                        if (zmq_std::send_recieve_wait(token_down, reply_down, node_socket) and
reply_down.action == success) {
                                *reply = reply_down;
                        }
                }
        }
        zmq_std::send_msg_dontwait(reply, node_parent_socket);
    }
    if (has_child) {
        rc = zmq_close(node_socket);
        assert(rc == 0);
        rc = zmq_ctx_term(node_context);
        assert(rc == 0);
    }
    rc = zmq_close(node_parent_socket);
    assert(rc == 0);
    rc = zmq_ctx_term(node_parent_context);
    assert(rc == 0);
}
```

control.cpp

```
#include <unistd.h>
#include <vector>
#include <limits>

#include "topology.hpp"
#include "zmq_std.hpp"

using node_id_type = long long;

int main() {
    int rc;
    topology_t<node_id_type> control_node;
    std::vector< std::pair<void*, void*> > childs;

    std::string s;
    node_id_type id;
    while (std::cin >> s) {
        if (s == "create") {
            node_id_type parent_id;
            std::cin >> id >> parent_id;
            if (parent_id == -1) {
                void* new_context = NULL;
                void* new_socket = NULL;
                zmq_std::init_pair_socket(new_context, new_socket);
                rc = zmq_bind(new_socket, ("tcp://*:" +
std::to_string(PORT_BASE + id)).c_str());
                assert(rc == 0);
```

```
                                int fork_id = fork();
                                if (fork_id == 0) {
                                        rc = execl(NODE_EXECUTABLE_NAME, NODE_EXECUTABLE_NAME,
std::to_string(id).c_str(), NULL);
                                        assert(rc != -1);
                                        return 0;
                                } else {
                                        bool ok = true;
                                        node_token_t reply_info({fail, id, id});
                                        ok = zmq_std::recieve_msg_wait(reply_info, new_socket);

                                        node_token_t* token = new node_token_t({ping, id, id});
                                        node_token_t reply({fail, id, id});
                                        ok = zmq_std::send_recieve_wait(token, reply,
new_socket);
                                        if (ok and reply.action == success) {
                                                childs.push_back(std::make_pair(new_context,
new_socket));
                                                control_node.insert(id);
                                                std::cout << "OK: " << reply_info.id <<
std::endl;
                                        } else {
                                                rc = zmq_close(new_socket);
                                                assert(rc == 0);
                                                rc = zmq_ctx_term(new_context);
                                                assert(rc == 0);
                                        }
                                }
                        } else if (control_node.find(parent_id) == -1) {
                                std::cout << "Error: Not found" << std::endl;
                        } else {
                                if (control_node.find(id) != -1) {
                                        std::cout << "Error: Already exists" << std::endl;
                                } else {
                                        int ind = control_node.find(parent_id);
                                        node_token_t* token = new node_token_t({create,
parent_id, id});
                                        node_token_t reply({fail, id, id});
                                        if (zmq_std::send_recieve_wait(token, reply,
childs[ind].second) and reply.action == success) {
                                                std::cout << "OK: " << reply.id << std::endl;
                                                control_node.insert(parent_id, id);
                                        } else {
                                                std::cout << "Error: Parent is unavailable" <<
std::endl;
                                        }
                                }
                        }
                } else if (s == "pingall") {
                        bool allprocess_flag = true;
                        std::vector<node_id_type> active_node;
                        for (node_id_type id : control_node.getAllPointers()) {
                  int ind = control_node.find(id);
                  node_token_t* token = new node_token_t({ping, id, id});
                                node_token_t reply({fail, id, id});
                                if (zmq_std::send_recieve_wait(token, reply,
childs[ind].second) and reply.action == success) {
                                        active_node.push_back(id);
```

```
                        } else {
                                allprocess_flag = false;
                        }
                }
                        if (allprocess_flag) {
                                std::cout << "OK: -1" << std::endl;
                        } else {
                                std::cout << "OK: ";
                                for (auto elem: active_node) {
                                        std::cout << elem << " ";
                                }
                                std::cout << std::endl;
                        }
                } else if (s == "exec") {
                        std::string key, value, res;
                        std::cin >> id >> key >> value;
                        int ind = control_node.find(id);
                        if (ind != -1) {
                                bool ok = true, ans = false;
                                int responce;
                                res = key + SENTINEL + value + SENTINEL;
                                for (size_t i = 0; i < res.size(); ++i) {
                                        node_token_t* token = new node_token_t({exec, res[i],
id});
                                        node_token_t reply({fail, id, id});
                                        if (!zmq_std::send_recieve_wait(token, reply,
childs[ind].second) or reply.action != success) {
                                                ok = false;
                                                if (reply.action == notfound) {
                                                        std::cout << "Error: Invalid key" <<
std::endl;
                                                        ans = true;
                                                }
                                                break;
                                        }
                                        if (i == res.size() - 1) {
                                                responce = reply.parent_id;
                                        }
                                }
                                if (ok && !ans) {
                                        std::cout << "OK: " << id;
                                        if (value == "get") {
                                                std::cout  <<  ":" << responce << std::endl;
                                        } else {
                                                std::cout << std::endl;
                                        }

                                } else if (!ans){
                                        std::cout << "Error: Node is unavailable" << std::endl;
                                }
                        } else {
                                std::cout << "Error: Not found" << std::endl;
                        }
                }
        }

        for (size_t i = 0; i < childs.size(); ++i) {
                rc = zmq_close(childs[i].second);
                assert(rc == 0);
```

```
                rc = zmq_ctx_term(childs[i].first);
                assert(rc == 0);
        }
}
```

Пример работы

create 1 -1

OK: 16205

create 2 1

OK: 16233

create 3 -1

OK: 16238

pingall

OK: -1

exec 1 a 1

OK: 1

exec 1 a get

OK: 1:1

exec 2 a get

Error: Invalid key

exec 2 a 5

OK: 2

Вывод

Во время выполнения этой лабораторной работы я узнал что существует технологии очередей сообщений, изучил основные понятия в zmq, долго разбирался с сокетами и адресами. Очередь сообщений я нахожу наиболее удобным способом взаимодействия процессов из всех рассмотренных на курсе.

Видимо поэтому они и используются при коммерческой разработке. Наиболее сложным для меня оказалось понять, какая именно топология от меня требуется и как её сделать.