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: Румынина Екатерина Александровна
Группа: М8О-201Б-21
Вариант: 8
Преподаватель: Миронов Евгений Сергеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2022

Содержание

Репозиторий

https://github.com/KetRum0/mai_os_labs

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 8:

- Топология 1 - все вычислительные узлы находятся в списке. Есть только один управляющий узел. Чтобы добавить новый вычислительный узел к управляющему, то необходимо выполнить команду: create id -1
- Тип 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 2 - Формат команды: ping id Команда проверяет доступность конкретного узла. Если узла нет, то необходимо выводить ошибку: «Error: Not found»

Общие сведения о программе

client.cpp – программа управляющего узла

server.cpp - программа вычислительного узла

topology.hpp – объявление функций для реализации топологии (список)

topology.cpp – программа для реализации топологии (список)

zmqf.hpp – объявление функций для работы с zmq

zmqf.cpp – программа для реализации функций для работы с zmq

Исходный код

client.cpp

```
#include <unistd.h>
#include <iostream>
#include <vector>
#include <zmq.hpp>
#include <sstream>
#include "topology.h"
#include "zmqf.h"

int main() {
    Topology network;
    std::vector<zmq::socket_t> branches;
    zmq::context_t context;
    std::string command;
    zmq::socket_t main_socket(context, ZMQ_REP);
    std::string message;
    std::cout << "> ";

    while (std::cin >> command) {
        if (command == "create") {
            int node_id, parent_id;
            std::cin >> node_id >> parent_id;
            if (network.Find(node_id) != -1) {
                std::cout << "Error: already exists!\n";
            } else if (parent_id == -1) {
                pid_t pid = fork();
                if (pid < 0) {
                    perror("Can't create new process!\n");
                    exit(EXIT_FAILURE);
                }
                if (pid == 0) {
                    execl("server", "server",
std::to_string(node_id).c_str(), NULL);
                    perror("Can't execute new process!\n");
                    exit(EXIT_FAILURE);
                }
```

```
                branches.emplace_back(context, ZMQ_REQ);
                branches[branches.size() - 1].setsockopt(ZMQ_SNDTIMEO, 5000);
                bind(branches[branches.size() - 1], node_id);
                send_message(branches[branches.size() - 1],
std::to_string(node_id) + "pid");
                std::string reply = receive_message(branches[branches.size()
- 1]);
                std::cout << reply << "\n";
                network.Insert(node_id, parent_id);
            } else if (network.Find(parent_id) == -1) {
                std::cout << "Error: parent not found!\n";
            } else {
                int branch = network.Find(parent_id);
                send_message(branches[branch], std::to_string(parent_id) + "
create " + std::to_string(node_id));
                std::string reply = receive_message(branches[branch]);
                std::cout << reply << "\n";
                network.Insert(node_id, parent_id);
            }

        } else if (command == "remove") {
            int id;
            std::cin >> id;
            int branch = network.Find(id);
            if (branch == -1) {
                std::cout << "Error: incorrect node id!\n";
            } else {
                bool is_first = (network.GetFirstId(branch) == id);
                send_message(branches[branch], std::to_string(id) + "
remove");
                std::string reply = receive_message(branches[branch]);
                std::cout << reply << std::endl;
                network.Erase(id);
                if (is_first) {
                    unbind(branches[branch], id);
                    branches.erase(std::next(branches.begin(), branch));
                }
            }
```

```
        } else if (command == "exec") {
            size_t count = 0;
            int destId;
            std::cin >> destId;
            int branch = network.Find(destId);
            if (branch == -1) {
                std::cout << "Error: incorrect node id!\n";
            } else {
                std::string subcommand;
                std::cin >> subcommand;
                send_message(branches[branch], std::to_string(destId) + "exec
" + subcommand);
                std::string reply = receive_message(branches[branch]);
                std::cout << reply << "\n";
            }

        } else if (command == "ping") {
            int destId;
            std::cin >> destId;
            int branch = network.Find(destId);
            if (branch == -1) {
                std::cout << "Error: incorrect node id!\n";
            } else {
                send_message(branches[branch], std::to_string(destId) +
"ping");
                std::string reply = receive_message(branches[branch]);
                std::cout << reply << "\n";
            }

        } else if (command == "exit") {
            for (size_t i = 0; i < branches.size(); ++i) {
                int first_node_id = network.GetFirstId(i);
                send_message(branches[i], std::to_string(first_node_id) + "
remove");
                std::string reply = receive_message(branches[i]);
                if (reply != "OK") {
                    std::cout << reply << "\n";
```

```
                } else {
                    unbind(branches[i], first_node_id);
                }
            }
            exit(0);

        } else {
            std::cout << "Incorrect command: " << command << "<!\n";
        }

        std::cout << "> ";

    }
}
```

server.cpp

```
#include <unistd.h>
#include <iostream>
#include <string>
#include <vector>
#include <chrono>
#include "zmqf.h"
using namespace std::chrono;

int main(int argc, char* argv[])
{
    if (argc != 2 && argc != 3) {
        std::cout << "Wrong arguments. Not enough parameters!\n";
        exit(1);
    }
    int current_id = std::atoi(argv[1]);
    int child_id = -1;
    if (argc == 3) {
        child_id = std::atoi(argv[2]);
    }
    zmq::context_t context;
    zmq::socket_t parent_socket(context, ZMQ_REP);
    connect(parent_socket, current_id);
```

```
    zmq::socket_t child_socket(context, ZMQ_REQ);
    child_socket.setsockopt(ZMQ_SNDTIMEO, 5000);
    parent_socket.setsockopt(ZMQ_SNDTIMEO, 5000);
    std::string message;
    int flag = 0;
    uint64_t start;
    uint64_t stop;
    while (1) {
        zmq::message_t message_main;
        message = receive_message(parent_socket);
        //std::cout << message;
        std::string recieved_message(static_cast<char*>(message_main.data()),
message_main.size());
        //std::cout << recieved_message;
        std::istringstream request(message);
        int dest_id;
        request >> dest_id;
        std::string command;
        request >> command;
        if (dest_id == current_id) {
            if (command == "ping") {
                std::string ans = std::to_string(current_id) + ": Ok";
                send_message(parent_socket, ans);

            } else if (command == "pid") {
                send_message(parent_socket, "OK: " +
std::to_string(getpid()));

            } else if (command == "create") {
                    int new_child_id;
                    request >> new_child_id;
                    //std::cout << new_child_id;
                    if (child_id != -1) {
                        unbind(child_socket, child_id);
                    }
                    bind(child_socket, new_child_id);
                    pid_t pid = fork();
                    if (pid < 0) {
```

```
                        perror("Can't create new process!\n");
                        exit(1);
                    }
                    if (pid == 0) {
                        execl("server", "server",
std::to_string(new_child_id).c_str(), std::to_string(child_id).c_str(),
NULL);
                        perror("Can't create new process!\n");
                        exit(1);
                    }
                    send_message(child_socket, std::to_string(new_child_id) +
"pid");
                    child_id = new_child_id;
                    send_message(parent_socket,
receive_message(child_socket));

            } else if (command == "remove") {
                    send_message(parent_socket, "OK");
                    disconnect(parent_socket, current_id);
                    break;
            } else if (command == "exec") {
                    std::string msg = "OK: " + std::to_string(dest_id) + " ";
                    std::string subcommand;
                    request >> subcommand;
                    if (subcommand == "start"){
                        if(flag == 1){
                            msg += "timer is already started";
                        }
                        else{
                            flag = 1;
                            msg += "started";
                            start =
duration_cast<milliseconds>(system_clock::now().time_since_epoch()).count();
                        }
                    } else if(subcommand == "stop") {
                        if(flag == 0) {
                            msg += "timer is not started";
                        }
```

```
                        else {
                            flag = 0;
                            msg += "stopped";
                        }
                    } else if(subcommand == "time") {
                        if(flag == 0) {
                            msg += "0";
                        }
                        else {
                            stop =
duration_cast<milliseconds>(system_clock::now().time_since_epoch()).count();
                            msg += std::to_string(stop-start);
                        }
                    } else {
                        msg += "Error: incorrect subcommand!\n";
                    }
                    send_message(parent_socket, msg);
                }
            } else if (child_id != -1) {
                send_message(child_socket, message);
                send_message(parent_socket, receive_message(child_socket));
                if (child_id == dest_id && command == "remove") {
                    child_id = -1;
                }
            } else {
                send_message(parent_socket, "Error: node is unavailable!\n");
            }
        }
}
```

topology.hpp

```
#ifndef LAB6_TOPOLOGY_H
#define LAB6_TOPOLOGY_H

#include <list>
#include <stdexcept>

class Topology {
```

```
private:
    std::list<std::list<int>> container;

public:
    void Insert(int id, int parent_id);
    int Find(int id);
    void Erase(int id);
    int GetFirstId(int list_id);
};

#endif // LAB6_TOPOLOGY_H
```

topology.cpp

```
#include <topology.h>

void Topology::Insert(int id, int parent_id) {
    if (parent_id == -1) {
        std::list<int> new_list;
        new_list.push_back(id);
        container.push_back(new_list);
    } else {
        int list_id = Find(parent_id);
        if (list_id == -1) {
            throw std::runtime_error("Insert Error: Wrong parent id");
        }
        auto it1 = container.begin();
        std::advance(it1, list_id);
        for (auto it2 = it1->begin(); it2 != it1->end(); ++it2) {
            if (*it2 == parent_id) {
                it1->insert(++it2, id);
                return;
            }
        }
    }
}

int Topology::Find(int id) {
    int cur_list_id = 0;
```

```
    for (auto it1 = container.begin(); it1 != container.end(); ++it1) {
        for (auto it2 = it1->begin(); it2 != it1->end(); ++it2) {
            if (*it2 == id) {
                return cur_list_id;
            }
        }
        ++cur_list_id;
    }
    return -1;
}

void Topology::Erase(int id) {
    int list_id = Find(id);
    if (list_id == -1) {
        throw std::runtime_error("Erase Error: Wrong id");
    }
    auto it1 = container.begin();
    std::advance(it1, list_id);
    for (auto it2 = it1->begin(); it2 != it1->end(); ++it2) {
        if (*it2 == id) {
            it1->erase(it2, it1->end());
            if (it1->empty()) {
                container.erase(it1);
            }
            return;
        }
    }
}

int Topology::GetFirstId(int list_id) {
    auto it1 = container.begin();
    std::advance(it1, list_id);
    if (it1->begin() == it1->end()) {
        return -1;
    }
    return *(it1->begin());
}
```

zmqf.hpp

```
#ifndef LAB6_ZMQ_F_H
#define LAB6_ZMQ_F_H

#include <iostream>
#include <string>
#include <zmq.hpp>

const int MAIN_PORT = 4040;

void send_message(zmq::socket_t& socket, const std::string& msg);

std::string receive_message(zmq::socket_t& socket);

void connect(zmq::socket_t& socket, int id);

void disconnect(zmq::socket_t& socket, int id);

void bind(zmq::socket_t& socket, int id);

void unbind(zmq::socket_t& socket, int id);

#endif
```

zmqf.cpp

```
#include "zmqf.h"

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) {
        return "Error";
    }
    std::string received_msg(static_cast<char*>(message.data()),
message.size());
    return received_msg;
}

void connect(zmq::socket_t& socket, int id) {
    std::string adress = "tcp://127.0.0.1:" + std::to_string(MAIN_PORT + id);
    socket.connect(adress);
}

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

void bind(zmq::socket_t& socket, int id) {
    std::string adress = "tcp://127.0.0.1:" + std::to_string(MAIN_PORT + id);
    socket.bind(adress);
}

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

Демонстрация работы программы

```
ket@ket-laptop:~/Desktop/mai_os_labs/lab678$ ./client
> create 1 -1
OK: 13687
> create 2 1
OK: 13690
> create 3 2
```

```
OK: 13693
> create 4 3
OK: 13696
> ping 1
1: Ok
> ping 4
4: Ok
> exec 1 start
OK: 1 started
> exec 2 start
OK: 2 started
> exec 1 time
OK: 1 12419
> exec 2 time
OK: 2 10666
> exec 2 stop
OK: 2 stopped
> exec 2 time
OK: 2 0
> remove 3
OK
> ping 3
Error: incorrect node id!
> ping 4
Error: incorrect node id!
> ping 2
2: Ok
> exit
```

Выводы

В результате выполнения данной лабораторной работы я приобрела практические навыки в управлении серверами сообщений, применении отложенных вычислений, интеграции программных систем друг с другом.