Московский авиационный институт
(Национальный исследовательский университет)
Факультет "Информационные технологии и прикладная математика"
Кафедра "Вычислительная математика и программирование"

Лабораторная работа №5-7 по курсу “Операционные системы”

Студент: Былькова Кристина Алексеевна

Группа: М8О-208Б-22

Преподаватель: Миронов Евгений Сергеевич

Вариант: 43

Оценка: _____

Дата: _____

Подпись: _____

Москва, 2023

Содержание

1 Репозиторий

https://github.com/kr1st1na0/OS_labs

2 Цель работы

Приобретение практических навыков в:

- Управлении серверами сообщений
- Применении отложенных вычислений
- Интеграции программных систем друг с другом

3 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.

4 Описание работы программы

Топология 2:

Все вычислительные узлы находятся в дереве общего вида. Есть только один управляющий узел. Чтобы добавить новый вычислительный узел к управляющему, то необходимо выполнить команду: create id -1.

Набор команд 1:

Подсчет суммы n чисел - формат команды: exec id n k_1 ... k_n id – целочисленный идентификатор вычислительного узла, на который отправляется команда n – количество складываемых чисел, k_1 ... k_n – складываемые числа.

Команда проверки 2:

Формат команды: ping id Команда проверяет доступность конкретного узла. Если узла нет, то необходимо выводить ошибку: «Error: Not found».

В ходе выполнения лабораторной работы я использовала библиотеку ZeroMQ и следующие команды:

- bind() - устанавливает "сокет"на адрес, а затем принимает входящие соединения на этом адресе.
- unbind() - отвязывает сокет от адреса
- connect() - создание соединения между сокетом и адресом
- disconnect() - разрывает соединение между сокетом и адресом
- send() - отправка сообщений
- recv() - получение сообщений

5 Исходный код

nodeRoutine.hpp

```
#pragma once

#include <iostream>
#include <sstream>
#include <unordered_map>
#include <optional>
#include <memory>
#include "unistd.h"

#include "socketRoutine.hpp"

class Node{
private:
    zmq::context_t context;
public:
    std::unordered_map<int, std::unique_ptr<zmq::socket_t> > children;
    std::unordered_map<int, int> childrenPort;
    int id;
    zmq::socket_t parent;
    int parentPort;

    Node(int _id, int _parentPort = -1): id(_id), parent(context, ZMQ_REP), parentPort(_parentPort) {
        if(_id != -1) {
            Connect(&parent, _parentPort);
        }
    }

    std::string Ping(int _id);
    std::string Create(int idChild, const std::string& programPath);
    std::string Pid();
    std::string Send(const std::string& str, int _id);
    std::string Kill();
};
```

socketRoutine.hpp

```
#pragma once

#include <iostream>
#include <sstream>
#include <string>
#include <optional>

#include <zmq.hpp>

int Bind(zmq::socket_t *socket, int id);
void Unbind(zmq::socket_t *socket, int port);

void Connect(zmq::socket_t *socket, int port);
void Disconnect(zmq::socket_t *socket, int port);

bool SendMessage(zmq::socket_t *socket, const std::string& msg);
std::optional<std::string> ReceiveMessage(zmq::socket_t *socket);
```

nodeRoutine.cpp

```cpp
#include "nodeRoutine.hpp"

std::string Node::Ping(int _id) {
    std::string ans = "Ok: 0";
    if (_id == id) {
        ans = "Ok: 1";
        return ans;
    } else if (auto it = children.find(_id); it != children.end())
    {
        std::string msg = "ping " + std::to_string(_id);
        SendMessage(it->second.get(), msg);
        if (auto msg = ReceiveMessage(children[_id].get()); msg.
    has_value(), msg == "Ok: 1") {
            ans = *msg;
        }
        return ans;
    }
    return ans;
}

std::string Node::Create(int idChild, const std::string&
    programPath) {
    std::string programName = programPath.substr(programPath.
    find_last_of("/") + 1);
    children[idChild] = std::make_unique<zmq::socket_t>(context,
    ZMQ_REQ);

    int newPort = Bind(children[idChild].get(), idChild);
    childrenPort[idChild] = newPort;
    int pid = fork();

    if (pid == 0) { // Child process
        execl(programPath.c_str(), programName.c_str(), std::
    to_string(idChild).c_str(), std::to_string(newPort).c_str(),
    nullptr);
    } else { // Parent process
        std::string pidChild = "Error: couldn't connect to child";
        children[idChild]->setsockopt(ZMQ_SNDTIMEO,  3000);
        SendMessage(children[idChild].get(), "pid");
        if (auto msg = ReceiveMessage(children[idChild].get());
    msg.has_value()) {
            pidChild = *msg;
        }
        return "Ok: " + pidChild;
    }
    return 0;
}

std::string Node::Pid() {
    return std::to_string(getpid());
}

std::string Node::Send(const std::string& str, int id) {
    if (children.size() == 0) {
        return "Error: Not found";
    } else if (auto it = children.find(id); it != children.end())
    {
        if (SendMessage(it->second.get(), str)) {
```

```
            std::string ans = "Error: Not found";
            if (auto msg = ReceiveMessage(children[id].get()); msg.has_value()) {
                ans = *msg;
            }
            return ans;
        }
    } else {
        std::string ans = "Error: Not found";
        for (auto& child : children) {
            std::string msg = "send " + std::to_string(id) + " " + str;
            if (SendMessage(child.second.get(), msg)) {
                if (auto msg = ReceiveMessage(child.second.get()); msg.has_value()) {
                    ans = *msg;
                }
            }
        }
        return ans;
    }
    return 0;
}

std::string Node::Kill() {
    std::string ans;
    for (auto& child : children) {
        std::string msg = "kill";
        if (SendMessage(child.second.get(), msg)) {
            if (auto tmp = ReceiveMessage(child.second.get()); tmp.has_value()) {
                msg = *tmp;
            }
            if (ans.size() > 0) {
                ans = ans + " " + msg;
            } else {
                ans =  msg;
            }
        }
        Unbind(child.second.get(), childrenPort[child.first]);
        child.second->close();
    }
    children.clear();
    childrenPort.clear();
    return ans;
}
```

socketRoutine.cpp

```
#include "socketRoutine.hpp"

int Bind(zmq::socket_t *socket, int id) {
    int port = 4040 + id;
    while(true) {
        std::string address = "tcp://127.0.0.1:"  + std::to_string(port);
        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);
}

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);
}

bool SendMessage(zmq::socket_t *socket, const std::string& msg) {
    zmq::message_t message(msg.size());
    memcpy(message.data(), msg.c_str(), msg.size());
    try {
        socket->send(message, 0);
        return true;
    } catch(...) {
        return false;
    }
}

std::optional<std::string> ReceiveMessage(zmq::socket_t *socket) {
    zmq::message_t message;
    socket->recv(&message, 0);
    std::string received(static_cast<char*>(message.data()), message.size());
    return received.empty() ? std::nullopt : std::make_optional(received);
}
```

server.cpp

```
#include "nodeRoutine.hpp"
#include "socketRoutine.hpp"
#include <fstream>
#include <signal.h>

int main(int argc, char **argv) {
    if (argc != 3) {
        perror("Not enough arguments");
        exit(EXIT_FAILURE);
    }

    Node task(atoi(argv[1]), atoi(argv[2]));
    std::string programPath = getenv("PROGRAM_PATH");
    while(1) {
        std::string message;
```

```
16          std::string command = " ";
17          if (auto msg = ReceiveMessage(&(task.parent)); msg.
    has_value()) {
18              message = *msg;
19          }
20          std::istringstream request(message);
21          request >> command;
22
23          if (command == "create") {
24              int idChild;
25              request >> idChild;
26              std::string ans = task.Create(idChild, programPath);
27              SendMessage(&task.parent, ans);
28          } else if (command == "pid") {
29              std::string ans = task.Pid();
30              SendMessage(&task.parent, ans);
31          } else if (command == "ping") {
32              int idChild;
33              request >> idChild;
34              std::string ans = task.Ping(idChild);
35              SendMessage(&task.parent, ans);
36          } else if (command == "send") {
37              int id;
38              request >> id;
39              std::string str;
40              getline(request, str);
41              str.erase(0, 1);
42              std::string ans;
43              ans = task.Send(str, id);
44              SendMessage(&task.parent, ans);
45          } else if (command == "exec") {
46              int cnt, sum = 0, number;
47              request >> cnt;
48              for(int i = 0; i < cnt; i++) {
49                  request >> number;
50                  sum += number;
51              }
52              std::string to_send;
53              to_send = "Ok: " + std::to_string(task.id) + ": " +
    std::to_string(sum);
54              SendMessage(&task.parent,to_send);
55          } else if (command == "kill") {
56              std::string ans = task.Kill();
57              ans = std::to_string(task.id) + " " + ans;
58              SendMessage(&task.parent, ans);
59              Disconnect(&task.parent, task.parentPort);
60              task.parent.close();
61              break;
62          }
63      }
64
65      return 0;
66  }
```

client.cpp

```
1  #include "set"
2
3  #include "nodeRoutine.hpp"
4  #include "socketRoutine.hpp"
```

```cpp

int main() {
    std::set<int> Nodes;
    std::string programPath = getenv("PROGRAM_PATH");
    Node task(-1);
    Nodes.insert(-1);
    std::string command;
    while (std::cin >> command) {
        if (command == "create") {
            int idChild, idParent;
            std::cin >> idChild >> idParent;
            if (Nodes.find(idChild) != Nodes.end()) {
                std::cout << "Error: Already exists" << std::endl;
            } else if (Nodes.find(idParent) == Nodes.end()) {
                std::cout << "Error: Parent not found" << std::endl;
            }else if (idParent == task.id) { // from -1
                std::string ans = task.Create(idChild, programPath);
                std::cout << ans << std::endl;
                Nodes.insert(idChild);
            } else { // from other node
                std::string str = "create " + std::to_string(idChild);
                std::string ans = task.Send(str, idParent);
                std::cout << ans << std::endl;
                Nodes.insert(idChild);
            }
        } else if (command == "ping") {
            int idChild;
            std::cin >> idChild;
            if (Nodes.find(idChild) == Nodes.end()) {
                std::cout << "Error: Not found" << std::endl;
            } else if (task.children.find(idChild) != task.children.end()) {
                std::string ans = task.Ping(idChild);
                std::cout << ans << std::endl;
            } else {
                std::string str = "ping " + std::to_string(idChild);
                std::string ans = task.Send(str, idChild);
                if (ans == "Error: Not found") {
                    ans = "Ok: 0";
                }
                std::cout << ans << std::endl;
            }
        }else if (command == "exec") {
            int id, number, count;
            std::cin >> id >> count;
            std::string msg = "exec " + std::to_string(count);
            for (int i = 0; i < count; i++) {
                std::cin >> number;
                msg += " " + std::to_string(number);
            }
            if (Nodes.find(id) == Nodes.end()) {
                std::cout << "Error: Not found" << std::endl;
            } else {
                std::string ans = task.Send(msg, id);
                std::cout << ans << std::endl;
```

```
            }
        }else if(command == "kill") {
            int id;
            std::cin >> id;
            std::string msg = "kill";
            if (Nodes.find(id) == Nodes.end()) {
                std::cout << "Error: Not found" << std::endl;
            } else {
                std::string ans = task.Send(msg, id);
                if (ans != "Error: Not found") {
                    std::istringstream ids(ans);
                    int tmp;
                    while(ids >> tmp) {
                        Nodes.erase(tmp);
                    }
                    ans = "Ok";
                    if(task.children.find(id) != task.children.end
    ()) {
                        Unbind(task.children[id].get(), task.
    childrenPort[id]);
                        task.children[id]->close();
                        task.children.erase(id);
                        task.childrenPort.erase(id);
                    }
                }
                std::cout << ans << std::endl;
            }
        } else if (command == "exit") {
            task.Kill();
            return 0;
        }
    }
}
```

6 Тесты

```
#include <gtest/gtest.h>

#include "set"

#include "nodeRoutine.hpp"
#include "socketRoutine.hpp"

TEST(FifthSeventhLabTests, PingTest) {
    std::string programPath = getenv("PROGRAM_PATH");
    std::set<int> Nodes;
    Node task(-1);
    Nodes.insert(-1);
    task.Create(1, programPath);
    Nodes.insert(1);

    std::string ans = task.Send("ping 1", 1);
    EXPECT_EQ(ans, "Ok: 1");

    ans = task.Send("ping 2", 2);
    EXPECT_EQ(ans, "Error: Not found");

    task.Kill();
}

TEST(FifthSeventhLabTests, ExecTest) {
    std::string programPath = getenv("PROGRAM_PATH");
    std::set<int> Nodes;
    Node task(-1);
    Nodes.insert(-1);
    task.Create(1, programPath);
    Nodes.insert(1);

    std::string ans = task.Send("exec 5 6 4 2 8 5", 1);
    EXPECT_EQ(ans, "Ok: 1: 25");

    task.Kill();
}

TEST(FifthSeventhLabTests, FullTest) {
    std::string programPath = getenv("PROGRAM_PATH");
    std::string ans;
    std::set<int> Nodes;
    Node task(-1);
    Nodes.insert(-1);
    task.Create(1, programPath);
    Nodes.insert(1);
    task.Send("create 2", 1);
    Nodes.insert(2);
    task.Send("create 3", 2);
    Nodes.insert(3);

    ans = task.Send("ping 1", 1);
    EXPECT_EQ(ans, "Ok: 1");
    ans = task.Send("ping 2", 2);
    EXPECT_EQ(ans, "Ok: 1");

    ans = task.Send("exec 3 1 2 3", 2);
```

```
58      EXPECT_EQ(ans, "Ok: 2: 6");
59
60      ans = task.Send("exec 5 1 1 1 1 1", 3);
61      EXPECT_EQ(ans, "Ok: 3: 5");
62
63      task.Kill();
64  }
65
66  int main(int argc, char *argv[]) {
67      std::cout << getenv("PROGRAM_PATH") << std::endl;
68      // bash: export PROGRAM_PATH="/home/kristinab/ubuntu_main/
        OS_labs/build/lab5-7/server"
69      testing::InitGoogleTest(&argc, argv);
70      return RUN_ALL_TESTS();
71  }
```

7 Консоль

```
kristinab@LAPTOP-SFU9B1F4:~/ubuntu_main/OS_labs/build/lab5-7$ ./client
create 1 -1
Ok: 28795
create 2 -1
Ok: 28801
create 3 1
Ok: 28807
ping 1
Ok: 1
ping 2
Ok: 1
ping 3
Ok: 1
kill 3
Ok
ping 3
Error: Not found
exec 1 5 1 2 3 4 1
Ok: 1: 11
kill 2
Ok
exit
```

8 Запуск тестов

```
kristinab@LAPTOP-SFU9B1F4:~/ubuntu_main/OS_labs/build/tests$ ./lab5-7_test
/home/kristinab/ubuntu_main/OS_labs/build/lab5-7/server
[==========] Running 3 tests from 1 test suite.
[----------] Global test environment set-up.
[----------] 3 tests from FifthSeventhLabTests
[ RUN      ] FifthSeventhLabTests.PingTest
[       OK ] FifthSeventhLabTests.PingTest (23 ms)
[ RUN      ] FifthSeventhLabTests.ExecTest
[       OK ] FifthSeventhLabTests.ExecTest (3 ms)
[ RUN      ] FifthSeventhLabTests.FullTest
[       OK ] FifthSeventhLabTests.FullTest (12 ms)
[----------] 3 tests from FifthSeventhLabTests (39 ms total)

[----------] Global test environment tear-down
[==========] 3 tests from 1 test suite ran. (39 ms total)
[  PASSED  ] 3 tests.
```

9 Выводы

В результате выполнения данной лабораторной работы была реализована распределенная система по асинхронной обработке запросов в соответствие с вариантом задания на C++. Я приобрела практические навыки в управлении серверами сообщений, применении отложенных вычислений и интеграции программных систем друг с другом.