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: Кочкожаров Иван Вячеславович

Группа: М8О-208Б-22

Преподаватель: Миронов Евгений Сергеевич

Вариант: 17

Оценка: _____

Дата: _____

Подпись: _____

Москва, 2023

Содержание

1 Репозиторий

https://github.com/kochkozharov/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 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.

Набор команд 4:

Поиск подстроки в строке.

Формат команды: exec id text_string pattern_string. result – номера позиций, где найден образец, разделенный точкой с запятой

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 использована библиотека 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

socket.cpp

```cpp
#include <socket.h>

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

static std::string GetAddress(int sockId) {
    constexpr int MAIN_PORT = 4000;
    return "tcp://127.0.0.1:" + std::to_string(MAIN_PORT + sockId);
}

Socket::Socket(zmq::socket_type t) : sock(ctx, t) {}

bool Socket::connect(int id) {
    try {
        sock.connect(GetAddress(id));
    } catch (...) {
        return false;
    }
    return true;
}

void Socket::disconnect(int id) { sock.disconnect(GetAddress(id)); }

void Socket::bind(int id) {
    try {
        sock.bind(GetAddress(id));
    } catch (...) {
        throw std::runtime_error("Error: port already in use");
    }
}

void Socket::unbind(int id) { sock.unbind(GetAddress(id)); }

bool Socket::sendMessage(const std::string &msg) {
    return sock.send(zmq::buffer(msg), zmq::send_flags::none).has_value();
}
std::optional<std::string> Socket::receiveMessage(bool nowait) {
    zmq::message_t zmsg{};
    auto len = sock.recv(
        zmsg, nowait ? zmq::recv_flags::dontwait : zmq::recv_flags::none);
    if (len) {
        return zmsg.to_string();
    }
    return {};
}
```

node.cpp

```cpp
#include "node.h"
```

```
2
3 #include "stdexcept"
4
5 ControlNode::ControlNode() : sock(zmq::socket_type::req) {}
6
7 ControlNode &ControlNode::get() {
8     static ControlNode instance;
9     return instance;
10 }
11
12 bool ControlNode::send(int id, const std::string &msg) {
13     auto status = sock.connect(id) && sock.sendMessage(msg);
14     return status;
15 }
16
17 std::optional<std::string> ControlNode::receive() {
18     return sock.receiveMessage(false);
19 }
20
21 ComputationNode::ComputationNode(int id) : sock(zmq::socket_type::
    rep), id(id) {
22     sock.bind(id);
23 }
24
25 ComputationNode::~ComputationNode() { sock.unbind(id); }
26
27 std::string ComputationNode::findAllOccurencies(const std::string
    &hay,
28                                                 const std::string
    &needle) {
29     std::string repMsg = "";
30     std::size_t pos = hay.find(needle, 0);
31     while (pos != std::string::npos) {
32         repMsg += std::to_string(pos) + ';';
33         pos = hay.find(needle, pos + 1);
34     }
35     repMsg.pop_back();
36     return repMsg;
37 }
38
39 void ComputationNode::computationLoop() {
40     while (true) {
41         auto reqMsg = sock.receiveMessage(false);
42         std::stringstream ss(reqMsg.value());
43         std::string command;
44         ss >> command;
45         if (command == "exec") {
46             std::string hay, needle;
47             ss >> hay >> needle;
48             sock.sendMessage("Ok: " + std::to_string(id) + ' ' +
    findAllOccurencies(hay, needle));
49         } else if (command == "ping") {
50             sock.sendMessage("pong");
51         }
52     }
53 }
```

topology.cpp

```
1 #include "topology.h"
```

```
2
3  Topology::NodeId::operator int() {
4      return id;
5  }
6
7  Topology::TopoIter Topology::begin() {
8      return {lists.begin(), lists.begin()->begin()};
9  }
10 Topology::TopoIter Topology::end() { return {lists.end(), lists.
       end()->end()}; }
11
12 bool Topology::insert(NodeId id, NodeId parentId) {
13     auto coord = find(id);
14     if (coord != end() || id == -1) {
15         std::cerr << "Error: Already exists\n";
16         return false;
17     }
18     if (parentId == -1) {
19         lists.insert(lists.end(), std::list(1, id));
20         return true;
21     }
22     auto coordParent = find(parentId);
23     if (coordParent == end()) {
24         std::cerr << "Error: Parent not found\n";
25         return false;
26     }
27     if (coordParent.it2 != (coordParent.it1)->end()) {
28         coordParent.it2++;
29     }
30     coordParent.it1->insert(coordParent.it2, id);
31     return true;
32 }
33
34 Topology::TopoIter Topology::find(NodeId id) {
35     auto it = begin();
36     for (size_t i = 0; i < lists.size(); ++i) {
37         it.it2 = it.it1->begin();
38         for (size_t j = 0; j < it.it1->size(); ++j) {
39             if (*it.it2 == id) {
40                 return it;
41             }
42             it.it2++;
43         }
44         it.it1++;
45     }
46     return end();
47 }
48
49 bool Topology::erase(NodeId id) {
50     auto coord = find(id);
51     if (coord == end() || id == -1) {
52         return false;
53     }
54     coord.it1->erase(coord.it2);
55     if (coord.it1->size() == 0) {
56         lists.erase(coord.it1);
57     }
58     return true;
59 }
```

```
60
61 bool operator==(Topology::TopoIter it1, Topology::TopoIter it2) {
62     return it1.it1 == it2.it1 && it1.it2 == it2.it2;
63 }
64
65 bool operator!=(Topology::TopoIter it1, Topology::TopoIter it2) {
66     return !(it1 == it2);
67 }
68
69 bool Topology::contains(NodeId id) { return find(id) != end(); }
```

client.cpp

```
1  #include <unistd.h>
2
3  #include <iostream>
4
5  #include "node.h"
6  #include "topology.h"
7   #include <signal.h>
8
9  int main(int argc, char *argv[]) {
10     if (argc != 2) {
11         std::cerr << "Not enough arguments\n";
12         std::exit(EXIT_FAILURE);
13     }
14     std::string command;
15     std::cout << "> ";
16
17     Topology topo;
18
19     while (std::cin >> command) {
20         if (command == "create") {
21             int id, parentId;
22             std::cin >> id >> parentId;
23
24             pid_t pid = fork();
25             if (pid == -1) {
26                 std::perror("fork");
27                 std::exit(EXIT_FAILURE);
28             }
29             if (pid == 0) {
30                 execl(argv[1], argv[1], std::to_string(id).c_str()
    ,
31                       std::to_string(parentId).c_str(), nullptr);
32
33             } else {
34                 if (!topo.insert(Topology::NodeId(id, pid),
    parentId)) {
35                     std::cout << "> ";
36                     std::cout.flush();
37                     continue;
38                 }
39                 std::cout << "Ok: " + std::to_string(pid) + '\n';
40             }
41         } else if (command == "ping") {
42             int id;
43             std::cin >> id;
44             if (!topo.contains(id)) {
45                 std::cerr << "Error: Not found\n";
```

```
                std::cout << "> ";
                std::cout.flush();
                continue;
            }
            if (!ControlNode::get().send(id, "ping")) {
                std::cout << "Ok: 0\n";
                std::cout << "> ";
                std::cout.flush();
                continue;
            }
            auto response = ControlNode::get().receive();
            if (response == "pong") {
                std::cerr << "Ok: 1\n";
            } else {
                std::cerr << "Ok: 0\n";
            }
        } else if (command == "exec") {
            int id;

            std::string hay, needle;
            std::cin >> id >> hay >> needle;
            if (!topo.contains(id)) {
                std::cerr << "Error: " + std::to_string(id) + "
    Not found\n";
                std::cout << "> ";
                std::cout.flush();
                continue;
            }
            if (!ControlNode::get().send(id, "exec " + hay + ' ' +
     needle)) {
                std::cerr << "Error: Node is unavailable";
                std::cout << "> ";
                std::cout.flush();
                continue;
            }
            auto response = ControlNode::get().receive();
            if (response) {
                std::cout << *response << '\n';
            }
        } else {
            std::cout << "Unknown command\n";
        }

        std::cout << "> ";
        std::cout.flush();
    }
    auto it = topo.begin();
    auto end = topo.end();
    for (;it.it1 != end.it1; it.it1++) {
        it.it2 = it.it1->begin();
        for (;it.it2 != it.it1->end(); it.it2++) {
            kill(it.it2->pid, SIGKILL);
            it.it2++;
        }
        it.it1++;
    }
}
```

server.cpp

```cpp
#include <node.h>
#include <iostream>


int main(int argc, char *argv[]) {
    if (argc != 3) {
        std::cerr << "Not enough arguments\n";
        std::exit(EXIT_FAILURE);
    }
    int id = std::stoi(argv[1]);
    ComputationNode cn(id);
    cn.computationLoop();
    return 0;
}
```

6 Тесты

lab5_test.cpp

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

#include "node.h"
#include "topology.h"

TEST(Lab5Tests, TopologyF) {
    Topology t;
    ASSERT_EQ(t.find(100), t.end());
    ASSERT_TRUE(t.insert(1, -1));
    ASSERT_EQ(t.find(1), t.begin());
    ASSERT_TRUE(t.insert(2, 1));
    auto it = t.begin();
    it.it2++;
    ASSERT_EQ(t.find(2), it);
    ASSERT_TRUE(t.insert(66, 2));
    it.it2++;
    ASSERT_EQ(t.find(66), it);
    ASSERT_TRUE(t.insert(3, -1));
    it.it1++;
    it.it2 = it.it1->begin();
    ASSERT_EQ(t.find(3), it);
    ASSERT_TRUE(t.insert(7, 3));
    it.it2++;
    ASSERT_EQ(t.find(7), it);
    ASSERT_FALSE(t.insert(2, -1));
    ASSERT_FALSE(t.insert(6, -4));
    ASSERT_EQ(t.find(100), t.end());
    ASSERT_TRUE(t.erase(1));
    ASSERT_EQ(t.find(1), t.end());
    ASSERT_EQ(t.find(2), t.begin());
    ASSERT_TRUE(t.erase(2));
    ASSERT_TRUE(t.erase(66));
    ASSERT_EQ(t.find(3), t.begin());
    ASSERT_TRUE(t.insert(2, 3));
    it = t.begin();
    it.it2++;
    ASSERT_EQ(t.find(2), it);
}

TEST(Lab5Tests, CalculationTest) {
    ASSERT_EQ(ComputationNode::findAllOccurencies("memmem", "mem")
    , "0;3");
    ASSERT_EQ(ComputationNode::findAllOccurencies("1000", "00"), "
    1;2");
}

TEST(Lab5Tests, ExecTest) {
    auto cstr = std::getenv("PATH_TO_SERVER");
    ASSERT_TRUE(cstr);
    pid_t pid = fork();
    if (pid == -1) {
        std::perror("fork");
        std::exit(EXIT_FAILURE);
    }
    if (pid == 0) {
```

```
55          execl(cstr, cstr, std::to_string(1).c_str(), std::
    to_string(-1).c_str(),
56                  nullptr);
57      }
58      ControlNode::get().send(1, "exec ooloo oo");
59      ASSERT_EQ(ControlNode::get().receive().value(), "Ok: 1 0;3");
60      kill(pid, SIGKILL);
61  }
```

7 Консоль

```
ivan@asus-vivobook ~/c/o/b/lab5_7 (reports) > client ./server
> create 1 -1
Ok: 78019
> create 2 1
Ok: 78038
> create 3 -1
Ok: 78060
> exec 3 memmem mem
Ok: 3 0;3
> ping 3
Ok: 1
> ping 4
Error: Not found
> exec 1 2222 2
Ok: 1 0;1;2;3
> exit
```

8 Запуск тестов

```
ivan@asus-vivobook ~/c/o/b/tests (reports) [SIGINT]> PATH_TO_SERVER=../lab5_7/serve
Running main() from /var/tmp/portage/dev-cpp/gtest-1.13.0/work/googletest-1.13.0/g
[==========] Running 3 tests from 1 test suite.
[----------] Global test environment set-up.
[----------] 3 tests from Lab5Tests
[ RUN      ] Lab5Tests.TopologyF
Error: Already exists
Error: Parent not found
[       OK ] Lab5Tests.TopologyF (0 ms)
[ RUN      ] Lab5Tests.CalculationTest
[       OK ] Lab5Tests.CalculationTest (0 ms)
[ RUN      ] Lab5Tests.ExecTest
[       OK ] Lab5Tests.ExecTest (127 ms)
[----------] 3 tests from Lab5Tests (128 ms total)

[----------] Global test environment tear-down
[==========] 3 tests from 1 test suite ran. (128 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++. Приобретены практические навыки в управлении серверами сообщений, применении отложенных вычислений и интеграции программных систем друг с другом.