

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: Сибирцев Роман Денисович

Группа: М8О-208Б-22

Преподаватель: Миронов Евгений Сергеевич

Вариант: 39

Оценка: _____

Дата: _____

Подпись: _____

Москва, 2023

Содержание

1	Репозиторий	3
2	Цель работы	3
3	Задание	3
4	Описание работы программы	3
5	Исходный код	4
6	Тесты	11
7	Запуск тестов	13
8	Выводы	14

1 Репозиторий

https://github.com/RomanSibirtsev/MAI_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`.

Набора команд 4 (поиск подстроки в строке) Формат команды: `> exec id; > text_string > pattern_string [result]` – номера позиций, где найден образец, разделенный точкой с запятой `text_string` — текст, в котором искать образец. Алфавит: `[A-Za-z0-9]`. Максимальная длина строки 108 символов `pattern_string` — образец 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

node.hpp

```
1 #pragma once
2
3 #include <iostream>
4 #include <sstream>
5 #include <unordered_map>
6 #include <optional>
7 #include <memory>
8 #include "unistd.h"
9
10 #include "socket.hpp"
11
12 class Node{
13 private:
14     zmq::context_t context;
15 public:
16     std::unordered_map<int, std::unique_ptr<zmq::socket_t> >
children;
17     std::unordered_map<int, int> childrenPort;
18     int id;
19     zmq::socket_t parent;
20     int parentPort;
21
22     Node(int _id, int _parentPort = -1): id(_id), parent(context,
ZMQ_REP), parentPort(_parentPort) {
23         if(_id != -1) {
24             Connect(&parent, _parentPort);
25         }
26     }
27
28     std::string Ping(int _id);
29     std::string Create(int idChild, const std::string& programPath
);
30     std::string Pid();
31     std::string Send(const std::string& str, int _id);
32     std::string Kill();
33 };
```

socket.hpp

```
1 #pragma once
2
3 #include <iostream>
4 #include <sstream>
5 #include <string>
6 #include <optional>
7
8 #include <zmq.hpp>
9
10 int Bind(zmq::socket_t *socket, int id);
11 void Unbind(zmq::socket_t *socket, int port);
12
13 void Connect(zmq::socket_t *socket, int port);
14 void Disconnect(zmq::socket_t *socket, int port);
15
16 bool SendMessage(zmq::socket_t *socket, const std::string& msg);
17 std::optional<std::string> ReceiveMessage(zmq::socket_t *socket);
```

node.cpp

```

1 #include "node.hpp"
2
3 std::string Node::Ping(int _id) {
4     std::string ans = "Ok: 0";
5     if (_id == id) {
6         ans = "Ok: 1";
7         return ans;
8     } else if (auto it = children.find(_id); it != children.end())
9     {
10         std::string msg = "ping " + std::to_string(_id);
11         SendMessage(it->second.get(), msg);
12         if (auto msg = ReceiveMessage(children[_id].get()); msg.
13             has_value(), msg == "Ok: 1") {
14             ans = *msg;
15         }
16         return ans;
17     }
18
19 std::string Node::Create(int idChild, const std::string&
20     programPath) {
21     std::string programName = programPath.substr(programPath.
22         find_last_of("/") + 1);
23     children[idChild] = std::make_unique<zmq::socket_t>(context,
24         ZMQ_REQ);
25
26     int newPort = Bind(children[idChild].get(), idChild);
27     childrenPort[idChild] = newPort;
28     int pid = fork();
29
30     if (pid == 0) { // Child process
31         execl(programPath.c_str(), programName.c_str(), std:::
32             to_string(idChild).c_str(), std::to_string(newPort).c_str(),
33             nullptr);
34     } else { // Parent process
35         std::string pidChild = "Error: couldn't connect to child";
36         children[idChild]->setsockopt(ZMQ_SNDTIMEO, 3000);
37         SendMessage(children[idChild].get(), "pid");
38         if (auto msg = ReceiveMessage(children[idChild].get());
39             msg.has_value()) {
40             pidChild = *msg;
41         }
42         return "Ok: " + pidChild;
43     }
44     return 0;
45 }
46
47 std::string Node::Pid() {
48     return std::to_string(getpid());
49 }
50
51 std::string Node::Send(const std::string& str, int id) {
52     if (children.size() == 0) {
53         return "Error: Not found";
54     } else if (auto it = children.find(id); it != children.end())
55     {
56         if (SendMessage(it->second.get(), str)) {

```

```

50         std::string ans = "Error: Not found";
51         if (auto msg = ReceiveMessage(children[id].get()); msg
.has_value()) {
52             ans = *msg;
53         }
54         return ans;
55     }
56 } else {
57     std::string ans = "Error: Not found";
58     for (auto& child : children) {
59         std::string msg = "send " + std::to_string(id) + " " +
str;
60         if (SendMessage(child.second.get(), msg)) {
61             if (auto msg = ReceiveMessage(child.second.get());
msg.has_value()) {
62                 ans = *msg;
63             }
64         }
65     }
66     return ans;
67 }
68 return 0;
69 }
70
71 std::string Node::Kill() {
72     std::string ans;
73     for (auto& child : children) {
74         std::string msg = "kill";
75         if (SendMessage(child.second.get(), msg)) {
76             if (auto tmp = ReceiveMessage(child.second.get()); tmp
.has_value()) {
77                 msg = *tmp;
78             }
79             if (ans.size() > 0) {
80                 ans = ans + " " + msg;
81             } else {
82                 ans = msg;
83             }
84         }
85         Unbind(child.second.get(), childrenPort[child.first]);
86         child.second->close();
87     }
88     children.clear();
89     childrenPort.clear();
90     return ans;
91 }

```

socket.cpp

```

1 #include "socket.hpp"
2
3 int Bind(zmq::socket_t *socket, int id) {
4     int port = 4040 + id;
5     while(true) {
6         std::string address = "tcp://127.0.0.1:" + std::to_string
(port);
7         try{
8             socket->bind(address);
9             break;
10        } catch(...) {

```

```

11         port++;
12     }
13 }
14 return port;
15 }
16 void Unbind(zmq::socket_t *socket, int port) {
17     std::string address = "tcp://127.0.0.1:" + std::to_string(port)
18 );
19     socket->unbind(address);
20 }
21 void Connect(zmq::socket_t *socket, int port) {
22     std::string address = "tcp://127.0.0.1:" + std::to_string(port)
23 );
24     socket->connect(address);
25 }
26 void Disconnect(zmq::socket_t *socket, int port) {
27     std::string address = "tcp://127.0.0.1:" + std::to_string(port)
28 );
29     socket->disconnect(address);
30 }
31 bool SendMessage(zmq::socket_t *socket, const std::string& msg) {
32     zmq::message_t message(msg.size());
33     memcpy(message.data(), msg.c_str(), msg.size());
34     try {
35         socket->send(message, 0);
36         return true;
37     } catch(...) {
38         return false;
39     }
40 }
41
42 std::optional<std::string> ReceiveMessage(zmq::socket_t *socket) {
43     zmq::message_t message;
44     socket->recv(&message, 0);
45     std::string received(static_cast<char*>(message.data()),
46 message.size());
47     return received.empty() ? std::nullopt : std::make_optional(
48 received);
49 }

```

control.cpp

```

1 #include "set"
2
3 #include "node.hpp"
4 #include "socket.hpp"
5
6 int main() {
7     std::set<int> Nodes;
8     std::string programPath = getenv("PROGRAM_PATH");
9     Node task(-1);
10    Nodes.insert(-1);
11    std::string command;
12    while (std::cin >> command) {
13        if (command == "create") {
14            int idChild, idParent;
15            std::cin >> idChild >> idParent;

```

```

16         if (Nodes.find(idChild) != Nodes.end()) {
17             std::cout << "Error: Already exists" << std::endl;
18         } else if (Nodes.find(idParent) == Nodes.end()) {
19             std::cout << "Error: Parent not found" << std::
endl;
20         } else if (idParent == task.id) { // from -1
21             std::string ans = task.Create(idChild, programPath
);
22             std::cout << ans << std::endl;
23             Nodes.insert(idChild);
24         } else { // from other node
25             std::string str = "create " + std::to_string(
idChild);
26             std::string ans = task.Send(str, idParent);
27             std::cout << ans << std::endl;
28             Nodes.insert(idChild);
29         }
30     } else if (command == "ping") {
31         int idChild;
32         std::cin >> idChild;
33         if (Nodes.find(idChild) == Nodes.end()) {
34             std::cout << "Error: Not found" << std::endl;
35         } else if (task.children.find(idChild) != task.
children.end()) {
36             std::string ans = task.Ping(idChild);
37             std::cout << ans << std::endl;
38         } else {
39             std::string str = "ping " + std::to_string(idChild
);
40             std::string ans = task.Send(str, idChild);
41             if (ans == "Error: Not found") {
42                 ans = "Ok: 0";
43             }
44             std::cout << ans << std::endl;
45         }
46     } else if (command == "exec") {
47         int id, number, count;
48         std::cin >> id >> count;
49         std::string msg = "exec " + std::to_string(count);
50         for (int i = 0; i < count; i++) {
51             std::cin >> number;
52             msg += " " + std::to_string(number);
53         }
54         if (Nodes.find(id) == Nodes.end()) {
55             std::cout << "Error: Not found" << std::endl;
56         } else {
57             std::string ans = task.Send(msg, id);
58             std::cout << ans << std::endl;
59         }
60     } else if (command == "kill") {
61         int id;
62         std::cin >> id;
63         std::string msg = "kill";
64         if (Nodes.find(id) == Nodes.end()) {
65             std::cout << "Error: Not found" << std::endl;
66         } else {
67             std::string ans = task.Send(msg, id);
68             if (ans != "Error: Not found") {
69                 std::istringstream ids(ans);

```



```

70         int tmp;
71         while(ids >> tmp) {
72             Nodes.erase(tmp);
73         }
74         ans = "Ok";
75         if(task.children.find(id) != task.children.end
    ()){
76             Unbind(task.children[id].get(), task.
childrenPort[id]);
77             task.children[id]->close();
78             task.children.erase(id);
79             task.childrenPort.erase(id);
80         }
81     }
82     std::cout << ans << std::endl;
83 }
84 } else if (command == "exit") {
85     task.Kill();
86     return 0;
87 }
88 }
89 }

```

calculate.cpp

```

1  #include "node.hpp"
2  #include "socket.hpp"
3  #include <fstream>
4  #include <signal.h>
5
6  int main(int argc, char **argv) {
7      if (argc != 3) {
8          perror("Not enough arguments");
9          exit(EXIT_FAILURE);
10     }
11
12     Node task(atoi(argv[1]), atoi(argv[2]));
13     std::string programPath = getenv("PROGRAM_PATH");
14     while(1) {
15         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         long unsigned int i, j;
47         std::string a, b, res = "";
48         request >> a >> b;
49         for (i = 0; i <= a.size() - b.size(); ++i) {
50             for (j = 0; j < b.size(); ++j) {
51                 if (a[i + j] != b[j]) {
52                     break;
53                 }
54             }
55             if (j == b.size()) {
56                 res += std::to_string(i) + ";";
57             }
58         }
59         if (res == "") res = "-1;";
60         std::string to_send;
61         to_send = "Ok: " + std::to_string(task.id) + ": " +
res;
62         SendMessage(&task.parent, to_send);
63     } else if (command == "kill") {
64         std::string ans = task.Kill();
65         ans = std::to_string(task.id) + " " + ans;
66         SendMessage(&task.parent, ans);
67         Disconnect(&task.parent, task.parentPort);
68         task.parent.close();
69         break;
70     }
71 }
72
73 return 0;
74 }

```

6 Тесты

```
1 #include <gtest/gtest.h>
2
3 #include "set"
4
5 #include "node.hpp"
6 #include "socket.hpp"
7
8 TEST(FifthSeventhLabTests, PingTest) {
9     std::string programPath = getenv("PROGRAM_PATH");
10    std::set<int> Nodes;
11    Node task(-1);
12    Nodes.insert(-1);
13    task.Create(1, programPath);
14    Nodes.insert(1);
15
16    std::string ans = task.Send("ping 1", 1);
17    EXPECT_EQ(ans, "Ok: 1");
18
19    ans = task.Send("ping 3", 3);
20    EXPECT_EQ(ans, "Error: Not found");
21
22    task.Kill();
23 }
24
25 TEST(FifthSeventhLabTests, ExecTest) {
26    std::string programPath = getenv("PROGRAM_PATH");
27    std::set<int> Nodes;
28    Node task(-1);
29    Nodes.insert(-1);
30    task.Create(1, programPath);
31    Nodes.insert(1);
32
33    std::string ans = task.Send("exec asdfgh asd", 1);
34    EXPECT_EQ(ans, "Ok: 1: 0;");
35
36    task.Kill();
37 }
38
39 TEST(FifthSeventhLabTests, FullTest) {
40    std::string programPath = getenv("PROGRAM_PATH");
41    std::string ans;
42    std::set<int> Nodes;
43    Node task(-1);
44    Nodes.insert(-1);
45    task.Create(1, programPath);
46    Nodes.insert(1);
47    task.Send("create 2", 1);
48    Nodes.insert(2);
49    task.Send("create 3", 2);
50    Nodes.insert(3);
51
52    ans = task.Send("ping 1", 1);
53    EXPECT_EQ(ans, "Ok: 1");
54    ans = task.Send("ping 2", 2);
55    EXPECT_EQ(ans, "Ok: 1");
56
57    ans = task.Send("exec qwqwqwqwqw qw", 2);
```

```
58     EXPECT_EQ(ans, "Ok: 2: 0;2;4;6;8;");
59
60     ans = task.Send("exec qwerty ert", 3);
61     EXPECT_EQ(ans, "Ok: 3: 2;");
62
63     task.Kill();
64 }
```

7 Запуск тестов

```
roman@DESKTOP-K3DH39N:~/MAI/MAI_OS_labs/build$ ./tests/lab5-7_test
Running main() from /home/roman/MAI/MAI_OS_labs/build/_deps/googletest-src/googletest
[=====] Running 3 tests from 1 test suite.
[-----] Global test environment set-up.
[-----] 3 tests from FifthSeventhLabTests
[ RUN      ] FifthSeventhLabTests.PingTest
[          OK ] FifthSeventhLabTests.PingTest (91 ms)
[ RUN      ] FifthSeventhLabTests.ExecTest
[          OK ] FifthSeventhLabTests.ExecTest (3 ms)
[ RUN      ] FifthSeventhLabTests.FullTest
[          OK ] FifthSeventhLabTests.FullTest (10 ms)
[-----] 3 tests from FifthSeventhLabTests (105 ms total)

[-----] Global test environment tear-down
[=====] 3 tests from 1 test suite ran. (105 ms total)
[ PASSED  ] 3 tests.
```

8 Выводы

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