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: Слетюрин Кирилл Сергеевич

Группа: М8О-208Б-22

Преподаватель: Миронов Евгений Сергеевич

Вариант: 26

Оценка: _____

Дата: _____

Подпись: _____

Москва, 2023

Содержание

1 Репозиторий

https://github.com/kirill483/OS

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 2:

Формат команды: exec id <Name> <value>.

Команда проверки 1:

Формат команды: pingall

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

calculationnode.cpp

```
#include "socket.hpp"
#include <string>

int main(int argc, char* argv[]) {
    if (argc != 2 && argc != 3) {
        throw std::runtime_error("Wrong args for counting node");
    }

    int curId = atoi(argv[1]);
    int childId = -1;
    if (argc == 3) {
        childId = atoi(argv[2]);
    }
    std::string adr;
    std::string path = getenv("PATH_TO_CLIENT");

    std::unordered_map<std::string, int> dictionary;

    zmq::context_t context;
    zmq::socket_t parentSocket(context, ZMQ_REP);

    connect(parentSocket, curId);

    zmq::socket_t childSocket(context, ZMQ_REQ);
    if (childId != -1) {
        bind(childSocket, childId);
    }
    childSocket.setsockopt(ZMQ_SNDTIMEO, 5000);

    std::string message;
    while (true) {
        message = receiveMessage(parentSocket);
        std::istringstream request(message);
        int destId;
        request >> destId;

        std::string command;
        request >> command;

        if (destId == curId) {

            if (command == "pid") {
                sendMessage(parentSocket, "OK: " + std::to_string(
    getpid()));
            } else if (command == "create") {

                int new_childId;
                request >> new_childId;
                if (childId != -1) {
                    unbind(childSocket, childId);
                }
                bind(childSocket, new_childId);
                pid_t pid = fork();
                if (pid < 0) {
                    std::cout << "Can't create new process" << std
    ::endl;
```

```
55                      return -1;
56                  }
57                  if (pid == 0) {
58                      if (execl(path.c_str(), path.c_str(), std::
    to_string(new_childId).c_str(), std::to_string(childId).c_str()
    , NULL)==-1) {
59                          std::cout << "Error with execl" << std::
    endl;
60                          perror("Error with execl");
61                          exit(EXIT_FAILURE);
62                      }
63                      std::cout << "Can't execute new process" <<
    std::endl;
64                      return -2;
65                  }
66                  sendMessage(childSocket, std::to_string(
    new_childId) + " pid");
67                  childId = new_childId;
68                  sendMessage(parentSocket, receiveMessage(
    childSocket));
69
70              } else if (command == "check") {
71                  std::string key;
72                  request >> key;
73                  if (dictionary.find(key) != dictionary.end()) {
74                      sendMessage(parentSocket, "OK: " + std::
    to_string(curId) + ": " + std::to_string(dictionary[key]));
75                  } else {
76                      sendMessage(parentSocket, "OK: " + std::
    to_string(curId) + ": '" + key + "' not found");
77                  }
78              } else if (command == "add") {
79                  std::string key;
80                  int value;
81                  request >> key >> value;
82                  dictionary[key] = value;
83                  sendMessage(parentSocket, "OK: " + std::to_string(
    curId));
84              }  else if (command == "ping") {
85                  std::string reply;
86                  if (childId != -1) {
87                      sendMessage(childSocket, std::to_string(
    childId) + " ping");
88                      std::string msg = receiveMessage(childSocket);
89                      reply += " " + msg;
90                  }
91                  sendMessage(parentSocket, std::to_string(curId) +
    reply);
92              } else if (command == "kill") {
93                  if (childId != -1) {
94                      sendMessage(childSocket, std::to_string(
    childId) + " kill");
95                      std::string msg = receiveMessage(childSocket);
96                      if (msg == "OK") {
97                          sendMessage(parentSocket, "OK");
98                      }
99                      unbind(childSocket, childId);
100                     disconnect(parentSocket, curId);
101                     break;
```

```
                }
                sendMessage(parentSocket, "OK");
                disconnect(parentSocket, curId);
                break;
            }
        }
         else if (childId != -1) {
            sendMessage(childSocket, message);
            sendMessage(parentSocket, receiveMessage(childSocket))
    ;
            if (childId == destId && command == "kill") {
                childId = -1;
            }
        } else {
            sendMessage(parentSocket, "Error: Node is unavailable"
    );
        }
    }
}
```

controlnode.cpp

```
#include "topology.hpp"
#include "socket.hpp"

int main() {

    std::string path = getenv("PATH_TO_CLIENT");
    Topology list;
    std::vector<zmq::socket_t> branches;
    std::set<int> not_available_nodes;
    zmq::context_t context;

    std::string command;

    while (true) {
        std::cin >> command;
        if (command == "create") {
            int nodeId, parentId;
            std::cin >> nodeId >> parentId;
            if (list.find(nodeId) != -1) {
                std::cout << "Error: Already exists" << std::endl;
            } else if (parentId == -1) {
                pid_t pid = fork();
                if (pid < 0) {
                    std::cout << "Can't create new process" << std
    ::endl;
                    return -1;
                } else if (pid == 0) {
                    //execl(path.c_str(), path.c_str(), std::
    to_string(nodeId).c_str(), NULL);
                    if (execl(path.c_str(), path.c_str(), std::
    to_string(nodeId).c_str(), NULL)==-1) {
                        std::cout << "Error with execl" << std::
    endl;
                        perror("Error with execl");
                        exit(EXIT_FAILURE);
                    }
                    std::cout << "Can't execute new process" <<
    std::endl;
```

```
34                      return -2;
35                  }
36                  branches.emplace_back(context, ZMQ_REQ);
37                  branches[branches.size() - 1].setsockopt(
    ZMQ_SNDTIMEO, 5000);
38                  bind(branches[branches.size()-1], nodeId);
39
40                  sendMessage(branches[branches.size() - 1], std::
    to_string(nodeId) + " pid");
41
42                  std::string reply = receiveMessage(branches[
    branches.size() - 1]);
43                  std::cout << reply << std::endl;
44                  list.insert(nodeId, parentId);
45
46              } else if (list.find(parentId) == -1) {
47
48                  std::cout << "Error: Parent not found" << std::
    endl;
49
50              } else {
51                  int branch = list.find(parentId);
52                  sendMessage(branches[branch], std::to_string(
    parentId) + "create " + std::to_string(nodeId));
53
54                  std::string reply = receiveMessage(branches[branch
    ]);
55                  std::cout << reply << std::endl;
56                  list.insert(nodeId, parentId);
57              }
58          } else if (command == "exec") {
59              std::string s;
60              getline(std::cin, s);
61              std::string execCommand;
62              std::vector<std::string> tmp;
63              std::string tmp1 = " ";
64              for (size_t i = 1; i < s.size(); i++) {
65                  tmp1 += s[i];
66                  if (s[i] == ' ' || i == s.size() - 1) {
67                      tmp.push_back(tmp1);
68                      tmp1 = " ";
69                  }
70              }
71              if (tmp.size() == 2) {
72                  execCommand = "check";
73              } else {
74                  execCommand = "add";
75              }
76              int destId = stoi(tmp[0]);
77              int branch = list.find(destId);
78              if (branch == -1) {
79                  std::cout << "There is no such node id" << std::
    endl;
80              } else {
81                  if (execCommand == "check") {
82                      sendMessage(branches[branch], tmp[0] + "check"
     + tmp[1]);
83
84                  } else if (execCommand == "add") {
```

```
85                      std::string value;
86                      sendMessage(branches[branch], tmp[0] + "add" +
     tmp[1] + " " + tmp[2]);
87                  }
88                  std::string reply = receiveMessage(branches[branch
    ]);
89                  std::cout << reply << std::endl;
90              }
91          } else if (command == "kill") {
92              int id;
93              std::cin >> id;
94              int branch = list.find(id);
95              if (branch == -1) {
96                  std::cout << " Error: incorrect node id" << std::
    endl;
97              } else {
98                  bool is_first = (list.getFirstId(branch) == id);
99                  sendMessage(branches[branch], std::to_string(id) +
     "kill");
100                 std::string reply = receiveMessage(branches[branch
    ]);
101                 std::cout << reply << std::endl;
102                 not_available_nodes.merge(list.getSetOfChilds(id))
    ;
103                 list.erase(id);
104                 if (is_first) {
105                     unbind(branches[branch], id);
106                     branches.erase(branches.begin() + branch);
107                 }
108             }
109         } else if (command == "pingall") {
110             for (size_t i = 0; i < branches.size(); ++i) {
111                 int first_node_id = list.getFirstId(i);
112                 sendMessage(branches[i], std::to_string(
    first_node_id) + " ping");
113
114                 std::string received_message = receiveMessage(
    branches[i]);
115                 std::istringstream reply(received_message);
116                 int node;
117                 while(reply >> node) {
118                  //   std::cout << "list " << node << std::endl;
119                     not_available_nodes.erase(node);
120                 }
121             }
122             if (not_available_nodes.empty()) {
123                 std::cout << "OK: -1" << std::endl;
124             } else {
125                 std::cout << "OK: ";
126                 for (size_t i : not_available_nodes) {
127                     std::cout << i << ' ';
128                 }
129                 std::cout << std::endl;
130             }
131         } else if (command == "exit") {
132             for (size_t i = 0; i < branches.size(); ++i) {
133                 int firstNodeId = list.getFirstId(i);
134                 sendMessage(branches[i], std::to_string(
    firstNodeId) + " kill");
```

```
                std::string reply = receiveMessage(branches[i]);
                if (reply != "OK") {
                    std::cout << reply << std::endl;
                } else {
                    unbind(branches[i], firstNodeId);
                }
            }
            exit(0);
        } else {
            std::cout << "Not correct command" << std::endl;
        }
    }
}
```

socket.cpp

```
#include <socket.hpp>

void sendMessage(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, zmq::send_flags::none);
}


std::string receiveMessage(zmq::socket_t& socket) {
    zmq::message_t msg;
    int msgReceiv;
    try {
        std::optional<size_t> result = socket.recv(msg);
        if (result) {
            msgReceiv = static_cast<int>(*result);
        }
    }
    catch (...) {
        msgReceiv = 0;
    }
    if (msgReceiv == 0) {
        return "Error: Node is unavailable";
    }
    std::string receivedMsg(static_cast<char*>(msg.data()), msg.
    size());
    return receivedMsg;

}

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

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

void bind(zmq::socket_t& socket, int id) {
    std::string address = "tcp://127.0.0.1:" + std::to_string(
```

```
    MAIN_PORT + id);
42      socket.bind(address);
43  }
44
45  void unbind(zmq::socket_t& socket, int id) {
46      std::string address = "tcp://127.0.0.1:" + std::to_string(
    MAIN_PORT + id);
47      socket.unbind(address);
48  }
```

6 Тесты

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

#include "topology.hpp"
#include "socket.hpp"
#include <thread>

TEST(FifthSeventhLabTest, SocketTest) {
    zmq::context_t context;
    zmq::socket_t repSocket(context, ZMQ_REP);
    bind(repSocket, 3);

    std::thread serverThread([&repSocket]() {
        std::string receivedMessage = receiveMessage(repSocket);
        EXPECT_EQ(receivedMessage, "TestMSG");

        sendMessage(repSocket, "ReplyMSG");
    });

    zmq::socket_t reqSocket(context, ZMQ_REQ);
    connect(reqSocket, 3);

    sendMessage(reqSocket, "TestMSG");

    std::string replyMessage = receiveMessage(reqSocket);
    EXPECT_EQ(replyMessage, "ReplyMSG");

    disconnect(reqSocket, 3);
    unbind(repSocket, 3);
    serverThread.join();
}

TEST(FifthSeventhLabTest, TopologyTest) {
    Topology topology;

    topology.insert(1, -1);
    topology.insert(2, 1);
    topology.insert(3, 2);

    EXPECT_EQ(topology.find(1), 0);
    EXPECT_EQ(topology.find(2), 0);
    EXPECT_EQ(topology.find(3), 0);

    EXPECT_EQ(topology.getFirstId(0), 1);

    topology.erase(2);

    EXPECT_EQ(topology.find(2), -1);
}


int main(int argc, char *argv[]) {
    testing::InitGoogleTest(&argc, argv);
    return RUN_ALL_TESTS();
}
```

7 Демонстрация работы программы

```
kirill@DESKTOP-7E05ERB:/mnt/c/Users/Kirill/OS/lab57/build$ export PATH_TO_CLIENT="/
kirill@DESKTOP-7E05ERB:/mnt/c/Users/Kirill/OS/lab57/build$ ./client
create 1 -1
OK: 1161
exec 1 F 10
OK: 1
exec 1 F
OK: 1: 10
pingall
OK: -1
exit
kirill@DESKTOP-7E05ERB:/mnt/c/Users/Kirill/OS/lab57/build$
```

8 Запуск тестов

```
kirill@DESKTOP-7E05ERB:/mnt/c/Users/Kirill/OS/lab57/build$ ./Test
[==========] Running 2 tests from 1 test suite.
[----------] Global test environment set-up.
[----------] 2 tests from FifthSeventhLabTest
[ RUN      ] FifthSeventhLabTest.SocketTest
[       OK ] FifthSeventhLabTest.SocketTest (1 ms)
[ RUN      ] FifthSeventhLabTest.TopologyTest
[       OK ] FifthSeventhLabTest.TopologyTest (0 ms)
[----------] 2 tests from FifthSeventhLabTest (1 ms total)

[----------] Global test environment tear-down
[==========] 2 tests from 1 test suite ran. (1 ms total)
[  PASSED  ] 2 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.