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		
	$\Gamma pynna:$	M8O-208B-22
Преподаватель:	Миронов Евген	ний Сергеевич
		<i>Вариант:</i> 43
	Оценка:	
	Дата:	
	Π од nuc ь:	

Содержание

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

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».

Bходе 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

```
1 #pragma once
3 #include <iostream>
4 #include <sstream>
5 #include <unordered_map>
6 #include <optional>
7 #include <memory>
8 #include "unistd.h"
10 #include "socketRoutine.hpp"
12 class Node{
13 private:
     zmq::context_t context;
15 public:
     std::unordered_map < int , std::unique_ptr < zmq::socket_t > >
     children;
     std::unordered_map<int, int> childrenPort;
      int id;
      zmq::socket_t parent;
19
     int parentPort;
20
      Node(int _id, int _parentPort = -1): id(_id), parent(context,
     ZMQ_REP), parentPort(_parentPort) {
          if(_id != -1) {
              Connect(&parent, _parentPort);
          }
      }
26
27
      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();
33 };
     socketRoutine.hpp
1 #pragma once
3 #include <iostream>
4 #include <sstream>
5 #include <string>
6 #include <optional>
8 #include <zmq.hpp>
int Bind(zmq::socket_t *socket, int id);
void Unbind(zmq::socket_t *socket, int port);
13 void Connect(zmq::socket_t *socket, int port);
14 void Disconnect(zmq::socket_t *socket, int port);
16 bool SendMessage(zmq::socket_t *socket, const std::string& msg);
17 std::optional < std::string > ReceiveMessage(zmq::socket_t *socket);
```

nodeRoutine.cpp

```
# #include "nodeRoutine.hpp"
3 std::string Node::Ping(int _id) {
      std::string ans = "Ok: O";
      if (_id == id) {
          ans = "0k: 1";
6
          return ans;
      } else if (auto it = children.find(_id); it != children.end())
          std::string msg = "ping " + std::to_string(_id);
9
          SendMessage(it->second.get(), msg);
          if (auto msg = ReceiveMessage(children[_id].get()); msg.
     has_value(), msg == "Ok: 1") {
              ans = *msg;
          }
          return ans;
      }
      return ans;
16
17 }
18
19 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,
21
     ZMQ_REQ);
22
      int newPort = Bind(children[idChild].get(), idChild);
      childrenPort[idChild] = newPort;
      int pid = fork();
26
      if (pid == 0) { // Child process
27
          execl(programPath.c_str(), programName.c_str(), std::
28
     to_string(idChild).c_str(), std::to_string(newPort).c_str(),
     nullptr);
      } else { // Parent process
29
          std::string pidChild = "Error: couldn't connect to child";
30
          children[idChild]->setsockopt(ZMQ_SNDTIMEO,
31
          SendMessage(children[idChild].get(), "pid");
          if (auto msg = ReceiveMessage(children[idChild].get());
     msg.has_value()) {
              pidChild = *msg;
34
          return "Ok: " + pidChild;
36
      }
37
38
      return 0;
39 }
40
41 std::string Node::Pid() {
      return std::to_string(getpid());
42
43 }
44
45 std::string Node::Send(const std::string& str, int id) {
      if (children.size() == 0) {
          return "Error: Not found";
47
      } else if (auto it = children.find(id); it != children.end())
48
          if (SendMessage(it->second.get(), str)) {
```

```
std::string ans = "Error: Not found";
50
               if (auto msg = ReceiveMessage(children[id].get()); msg
      .has_value()) {
                   ans = *msg;
               }
               return ans;
          }
      } else {
56
           std::string ans = "Error: Not found";
58
           for (auto& child : children) {
               std::string msg = "send " + std::to_string(id) + " " +
59
      str;
               if (SendMessage(child.second.get(), msg)) {
60
                   if (auto msg = ReceiveMessage(child.second.get());
      msg.has_value()) {
                        ans = *msg;
62
                   }
63
               }
64
          }
          return ans;
66
67
      }
      return 0;
69 }
70
  std::string Node::Kill() {
71
72
      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
76
      .has_value()) {
                   msg = *tmp;
               }
               if (ans.size() > 0) {
79
                   ans = ans + " " + msg;
80
               } else {
81
                   ans = msg;
               }
83
84
          Unbind(child.second.get(), childrenPort[child.first]);
85
86
           child.second->close();
87
      children.clear();
88
      childrenPort.clear();
89
      return ans;
90
91 }
     socketRoutine.cpp
1 #include "socketRoutine.hpp"
3 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
6
      (port);
          try{
               socket ->bind(address);
               break;
9
          } catch(...) {
```

```
port++;
          }
      }
      return port;
14
15 }
16 Void Unbind(zmq::socket_t *socket, int port) {
      std::string address = "tcp://127.0.0.1:" + std::to_string(port
     );
      socket ->unbind(address);
18
19 }
void Connect(zmq::socket_t *socket, int port) {
      std::string address = "tcp://127.0.0.1:" + std::to_string(port
      socket ->connect(address);
23
24 }
26 void Disconnect(zmq::socket_t *socket, int port) {
      std::string address = "tcp://127.0.0.1:" + std::to_string(port
28
      socket ->disconnect(address);
29 }
30
31 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 {
34
          socket ->send(message, 0);
35
          return true;
      } catch(...) {
          return false;
38
      }
39
40 }
41
42 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()),
45
     message.size());
      return received.empty() ? std::nullopt : std::make_optional(
46
     received);
47 }
     server.cpp
1 #include "nodeRoutine.hpp"
2 #include "socketRoutine.hpp"
3 #include <fstream>
4 #include <signal.h>
6 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) {
14
          std::string message;
```

```
std::string command = " ";
          if (auto msg = ReceiveMessage(&(task.parent)); msg.
     has_value()) {
               message = *msg;
18
19
          std::istringstream request(message);
          request >> command;
          if (command == "create") {
               int idChild;
               request >> idChild;
25
               std::string ans = task.Create(idChild, programPath);
26
               SendMessage(&task.parent, ans);
27
          } else if (command == "pid") {
               std::string ans = task.Pid();
29
               SendMessage(&task.parent, ans);
30
          } else if (command == "ping") {
               int idChild;
               request >> idChild;
               std::string ans = task.Ping(idChild);
               SendMessage(&task.parent, ans);
          } else if (command == "send") {
               int id;
               request >> id;
               std::string str;
               getline(request, str);
40
               str.erase(0, 1);
41
               std::string ans;
42
               ans = task.Send(str, id);
               SendMessage(&task.parent, ans);
          } else if (command == "exec") {
45
               int cnt, sum = 0, number;
               request >> cnt;
               for(int i = 0; i < cnt; i++) {
48
                   request >> number;
49
                   sum += number;
               }
               std::string to_send;
               to_send = "Ok: " + std::to_string(task.id) + ": " +
     std::to_string(sum);
               SendMessage(&task.parent,to_send);
          } else if (command == "kill") {
               std::string ans = task.Kill();
56
               ans = std::to_string(task.id) + " " + ans;
               SendMessage(&task.parent, ans);
               Disconnect(&task.parent, task.parentPort);
               task.parent.close();
60
               break;
          }
63
64
65
      return 0;
66 }
     client.cpp
1 #include "set"
3 #include "nodeRoutine.hpp"
4 #include "socketRoutine.hpp"
```

```
6 int main() {
      std::set<int> Nodes;
      std::string programPath = getenv("PROGRAM_PATH");
8
      Node task(-1);
Q
      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()) {
16
                   std::cout << "Error: Already exists" << std::endl;</pre>
17
               } else if (Nodes.find(idParent) == Nodes.end()) {
                   std::cout << "Error: Parent not found" << std::
19
     endl:
               }else if (idParent == task.id) { // from -1
20
                   std::string ans = task.Create(idChild, programPath
21
     );
                   std::cout << ans << std::endl;</pre>
                   Nodes.insert(idChild);
23
               } else { // from other node
                   std::string str = "create " + std::to_string(
25
     idChild);
                   std::string ans = task.Send(str, idParent);
26
                   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;</pre>
               } else if (task.children.find(idChild) != task.
35
     children.end()) {
                   std::string ans = task.Ping(idChild);
36
                   std::cout << ans << std::endl;</pre>
               } else {
38
                   std::string str = "ping " + std::to_string(idChild
     );
40
                   std::string ans = task.Send(str, idChild);
                   if (ans == "Error: Not found") {
41
                       ans = "0k: 0";
42
                   }
43
                   std::cout << ans << std::endl;</pre>
44
               }
45
          }else if (command == "exec") {
46
               int id, number, count;
               std::cin >> id >> count;
               std::string msg = "exec " + std::to_string(count);
49
               for (int i = 0; i < count; i++) {
                   std::cin >> number;
                   msg += " " + std::to_string(number);
               if (Nodes.find(id) == Nodes.end()) {
54
                   std::cout << "Error: Not found" << std::endl;</pre>
               } else {
56
                   std::string ans = task.Send(msg, id);
                   std::cout << ans << std::endl;</pre>
58
```

```
}
59
           }else if(command == "kill") {
60
               int id;
61
               std::cin >> id;
62
               std::string msg = "kill";
63
               if (Nodes.find(id) == Nodes.end()) {
                    std::cout << "Error: Not found" << std::endl;</pre>
65
               } else {
66
                    std::string ans = task.Send(msg, id);
                    if (ans != "Error: Not found") {
68
69
                        std::istringstream ids(ans);
                        int tmp;
70
                        while(ids >> tmp) {
71
                            Nodes.erase(tmp);
                        }
73
                        ans = "0k";
74
                        if(task.children.find(id) != task.children.end
      ()) {
                            Unbind(task.children[id].get(), task.
76
     childrenPort[id]);
77
                            task.children[id]->close();
                            task.children.erase(id);
                            task.childrenPort.erase(id);
79
                        }
80
                    }
                    std::cout << ans << std::endl;</pre>
82
               }
83
           } else if (command == "exit") {
84
               task.Kill();
               return 0;
86
           }
87
      }
88
89 }
```

6 Тесты

```
1 #include <gtest/gtest.h>
3 #include "set"
5 #include "nodeRoutine.hpp"
6 #include "socketRoutine.hpp"
8 TEST(FifthSeventhLabTests, PingTest) {
      std::string programPath = getenv("PROGRAM_PATH");
9
      std::set<int> Nodes;
      Node task(-1);
      Nodes.insert(-1);
      task.Create(1, programPath);
      Nodes.insert(1);
14
16
      std::string ans = task.Send("ping 1", 1);
      EXPECT_EQ(ans, "Ok: 1");
17
18
      ans = task.Send("ping 2", 2);
19
      EXPECT_EQ(ans, "Error: Not found");
20
21
      task.Kill();
22
23 }
24
_{\rm 25} TEST(FifthSeventhLabTests, ExecTest) {
      std::string programPath = getenv("PROGRAM_PATH");
      std::set<int> Nodes;
      Node task(-1);
     Nodes.insert(-1);
2.9
      task.Create(1, programPath);
31
      Nodes.insert(1);
32
      std::string ans = task.Send("exec 5 6 4 2 8 5", 1);
33
      EXPECT_EQ(ans, "Ok: 1: 25");
34
      task.Kill();
36
37 }
38
39 TEST(FifthSeventhLabTests, FullTest) {
      std::string programPath = getenv("PROGRAM_PATH");
40
      std::string ans;
41
      std::set<int> Nodes;
      Node task(-1);
43
      Nodes.insert(-1);
44
      task.Create(1, programPath);
45
      Nodes.insert(1);
46
      task.Send("create 2", 1);
47
      Nodes.insert(2);
48
      task.Send("create 3", 2);
49
      Nodes.insert(3);
50
      ans = task.Send("ping 1", 1);
52
      EXPECT_EQ(ans, "Ok: 1");
53
      ans = task.Send("ping 2", 2);
      EXPECT_EQ(ans, "Ok: 1");
56
      ans = task.Send("exec 3 1 2 3", 2);
57
```

```
EXPECT_EQ(ans, "Ok: 2: 6");
59
      ans = task.Send("exec 5 1 1 1 1 1", 3);
60
      EXPECT_EQ(ans, "Ok: 3: 5");
61
62
      task.Kill();
63
64 }
65
66 int main(int argc, char *argv[]) {
      std::cout << getenv("PROGRAM_PATH") << std::endl;</pre>
68
      // \ bash: \ export \ PROGRAM\_PATH = "/home/kristinab/ubuntu\_main/
      \textit{OS\_labs/build/lab5-7/server"}
      testing::InitGoogleTest(&argc, argv);
69
      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
0k
ping 3
Error: Not found
exec 1 5 1 2 3 4 1
Ok: 1: 11
kill 2
0k
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
          ] 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.