Московский Авиационный Институт
(Национальный Исследовательский Университет)
Факультет информационных технологий и прикладной математики
Кафедра вычислительной математики и программирования

Лабораторная работа №5-7 по курсу «Операционные системы»

Студент: Останина Анна Андреевна
Группа: М8О-208Б-22
Вариант: 7
Преподаватель: Миронов Евгений Сергеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2024

Содержание

Репозиторий

https://github.com/Imariiii/os_labs

Цель работы

Приобретение практических навыков в:

- Управлении серверами сообщений (№5)
- Применение отложенных вычислений (№6)
- Интеграция программных систем друг с другом (№7)

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. При убийстве («kill -9») любого вычислительного узла система должна пытаться максимально сохранять свою работоспособность, а именно все дочерние узлы убитого узла могут стать недоступными, но родительские узлы должны сохранить свою работоспособность.

Управляющий узел отвечает за ввод команд от пользователя и отправку этих команд на вычислительные узлы. Список основных поддерживаемых команд:

Создание нового вычислительного узла

Формат команды: create id [parent]

id – целочисленный идентификатор нового вычислительного узла

parent – целочисленный идентификатор родительского узла

Формат вывода:

«Ok: pid», где pid – идентификатор процесса для созданного вычислительного узла

«Error: Already exists» - вычислительный узел с таким идентификатором

уже существует

«Error: Parent not found» - нет такого родительского узла с таким идентификатором

«Error: Parent is unavailable» - родительский узел существует, но по каким-то причинам с ним не удается связаться

«Error: [Custom error]» - любая другая обрабатываемая ошибка

Пример:

> create 10 5

Ok: 3128

Исполнение команды на вычислительном узле

Формат команды: exec id [params]

id – целочисленный идентификатор вычислительного узла, на который отправляется команда

Формат вывода:

«Ok:id: [result]», где result – результат выполненной команды

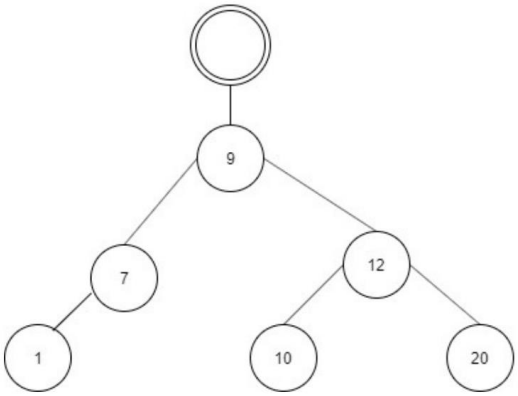
«Error:id: Not found» - вычислительный узел с таким идентификатором не найден

«Error:id: Node is unavailable» - по каким-то причинам не удается связаться с вычислительным узлом

«Error:id: [Custom error]» - любая другая обрабатываемая ошибка

Вариант 7

Топология 3



Все вычислительные узлы хранятся в бинарном дереве поиска. [parent] — является необязательным параметром.

Тип команд на вычислительных узлах

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 10^8 символов

pattern_string — образец

Пример:

> exec 10

> abracadabra

> abra

Ok:10:0;7

> exec 10

> abracadabra

> mmm

Ok:10: -1

Примечания: Выбор алгоритма поиска не важен

Тип проверки доступности узлов

Команда проверки 2

Формат команды: ping id

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

Пример:

> ping 10

Ok: 1 // узел 10 доступен

> ping 17

Ok: 0 // узел 17 недоступен

Описание работы программы

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

Исходный код

topology.h

```
#pragma once

#include <iostream>
#include <set>

struct NodeId {
    NodeId(int id) : id(id), pid(-1) {}
    NodeId(int id, int pid) : id(id), pid(pid) {}
    int id;
    int pid;
    operator int();
};

inline bool operator<(const NodeId& a, const NodeId& b) {
    return a.id < b.id;
}

using Topology = std::set<NodeId>;
```

socket.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
#include "node.h"

#include "stdexcept"

ControlNode::ControlNode() : sock(zmq::socket_type::req) {}

ControlNode &ControlNode::get() {
    static ControlNode instance;
    return instance;
}

bool ControlNode::send(int id, const std::string &msg) {
    auto status = sock.connect(id) && sock.sendMessage(msg);
    return status;
}

std::optional<std::string> ControlNode::receive() {
    return sock.receiveMessage(false);
}

ComputationNode::ComputationNode(int id) : sock(zmq::socket_type::rep),
id(id) {
    sock.bind(id);
}
```

```
ComputationNode::~ComputationNode() { sock.unbind(id); }

std::string ComputationNode::findAllOccurencies(const std::string &hay,
                                                const std::string &needle) {
    std::string repMsg = "";
    std::size_t pos = hay.find(needle, 0);
    while (pos != std::string::npos) {
        repMsg += std::to_string(pos) + ';';
        pos = hay.find(needle, pos + 1);
    }
    if (!repMsg.empty()) repMsg.pop_back();
    return repMsg;
}

void ComputationNode::computationLoop() {
    while (true) {
        auto reqMsg = sock.receiveMessage(false);
        std::stringstream ss(reqMsg.value());
        std::string command;
        ss >> command;
        if (command == "exec") {
            std::string hay, needle;
            ss >> hay >> needle;
            sock.sendMessage("Ok: " + std::to_string(id) + ' ' +
findAllOccurencies(hay, needle));
        } else if (command == "ping") {
            sock.sendMessage("pong");
        }
    }
}
```

client_util.cpp

```
#include <unistd.h>

#include <string>
#include <sstream>

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

bool ExecCommand(Topology &topo, const std::string &input,
                 const std::string &serverName) {
    std::stringstream ss(input);
    std::string command;
    ss >> command;
    if (command == "create") {
        int id, parentId;
        ss >> id >> parentId;
        if (topo.contains(id)) {
            std::cerr << "Error: Already exists\n";
            std::cout << "> ";
            std::cout.flush();
            return false;
        }
        pid_t pid = fork();
        if (pid == -1) {
            std::perror("fork");
            std::exit(EXIT_FAILURE);
        }
        if (pid == 0) {
            execl(serverName.c_str(), serverName.c_str(),
```

```
                    std::to_string(id).c_str(),
std::to_string(parentId).c_str(),
                    nullptr);

            } else {
                NodeId newNode(id, pid);
                topo.insert(newNode);
                std::cout << "Ok: " + std::to_string(pid) + '\n';
            }
        } else if (command == "ping") {
            int id;
            ss >> id;
            if (!topo.contains(id)) {
                std::cerr << "Error: Not found\n";
                std::cout.flush();
                return false;
            }
            if (!ControlNode::get().send(id, "ping")) {
                std::cout << "Ok: 0\n";
                std::cout.flush();
                return false;
            }
            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;
            ss >> id >> hay >> needle;
            if (!topo.contains(id)) {
                std::cerr << "Error: " + std::to_string(id) + " Not found\n";
                std::cout.flush();
                return false;
            }
            if (!ControlNode::get().send(id, "exec " + hay + ' ' + needle)) {
                std::cerr << "Error: Node is unavailable";
                std::cout.flush();
                return false;
            }
            auto response = ControlNode::get().receive();
            if (response) {
                std::cout << *response << '\n';
            }
        } else {
            std::cout << "Unknown command\n";
            return false;
        }
        return true;
}

client.cpp
#include <signal.h>
#include <unistd.h>

#include <iostream>

#include "node.h"
#include "topology.h"
#include "client_util.h"
```

```
int main(int argc, char *argv[]) {
    if (argc != 2) {
        std::cerr << "Not enough arguments\n";
        std::exit(EXIT_FAILURE);
    }
    std::string command;
    std::cout << "> ";

    Topology topo;
    while (std::getline(std::cin, command)) {
        ExecCommand(topo, command, argv[1]);
        std::cout << "> ";
        std::cout.flush();
    }

    for (auto node : topo) {
        kill(node.pid, SIGKILL);
    }
}
```

server.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;
}
```

Консоль

```
anna@anna-virtual-machine:~/labs_3sem/os_labs/build/lab5_7$ ./client server
> create 4 123
Ok: 34184
> ping 4
Ok: 1
> exec 4 qweqweqwe qwe
Ok: 4 0;3;6
> create 1 5636
Ok: 34394
> ping 1
Ok: 1
> exec 1 abracadabra abra
Ok: 1 0;7
```

Запуск тестов

```
anna@anna-virtual-machine:~/labs_3sem/os_labs/build/tests$
PATH_TO_SERVER=../lab5_7/server ./lab5_7_test
Running main() from /home/anna/labs_3sem/os_labs/build/_deps/googletest-
src/googletest/src/gtest_main.cc
[==========] Running 3 tests from 1 test suite.
[----------] Global test environment set-up.
[----------] 3 tests from Lab5Tests
```

```
[ RUN      ] Lab5Tests.CalculationTest
[       OK ] Lab5Tests.CalculationTest (0 ms)
[ RUN      ] Lab5Tests.ExecTest
[       OK ] Lab5Tests.ExecTest (21 ms)
[ RUN      ] Lab5Tests.TopoTest
Ok: 33794
Error: Already exists
> Ok: 33795
[       OK ] Lab5Tests.TopoTest (1 ms)
[----------] 3 tests from Lab5Tests (23 ms total)

[----------] Global test environment tear-down
[==========] 3 tests from 1 test suite ran. (23 ms total)
[  PASSED  ] 3 tests.
```

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.