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»

Тема работы
“Брокеры сообщений”

Студент: Слободин Никита Алексеевич
Группа: М8О-203Б-23
Вариант: 2

Преподаватель: Миронов Евгений Сергеевич

Оценка: _____
Дата: _____
Подпись: _____

Москва, 2024

Постановка задачи

Задача: 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

Топология: дерево общего вида.

Набор команд 4 (поиск подстроки в строке) Формат команды: exec id n k1 … kn id – целочисленный идентификатор вычислительного узла, на который отправляется команда n – количество складываемых чисел (от 1 до 108) k1 … kn – складываемые числа

Команда проверки 1 Формат команды: ping id Вывод ответа узла id на отправку сообщения

Общие сведения о программе

Программа реализует управление процессами и их взаимодействие через дерево узлов, используя технологии межпроцессного взаимодействия и архитектуру на основе ZeroMQ. Центральным элементом реализации является структура данных в виде общего дерева, представленного в tree. Каждый узел дерева хранит уникальный идентификатор, указатели на потомков, высоту и флаг доступности. Это дерево служит для логического отображения связей между процессами.

Главный процесс запускается в файле client.cpp. Он выполняет функции контроля за созданием, взаимодействием и мониторингом дочерних процессов. Создание новых процессов реализуется через системный вызов fork(), где дочерний процесс запускает отдельный экземпляр исполняемого файла node. В случае корневого узла выполняется прямой вызов процесса через execl(), а для других узлов отправляются сообщения родительскому процессу с указанием идентификатора нового узла.

Обмен сообщениями между процессами осуществляется с использованием библиотеки ZeroMQ.

Команды передаются в формате структур Message, содержащих тип сообщения и связанные с ним данные. Реализованы несколько типов сообщений: создание нового узла (create), выполнение команды (exec), проверка доступности узлов (ping), а также дополнительные сообщения для ответа и обработки ошибок. Для обработки команды exec дочерние процессы выполняют вычисления (например, суммирование чисел) и отправляют результаты через стандартный поток вывода.

Ключевым моментом является проверка доступности узлов через команду ping.

Реализация дочерних процессов (server.cpp) включает цикл обработки сообщений с помощью ZeroMQ. Каждый узел может создавать дочерние процессы, выполнять вычисления или подтверждать свою доступность в ответ на запросы.

Исходный код всех модулей представлен в приложении.

Выводы

В ходе выполнения работы была реализована система управления процессами с использованием общего дерева для организации логической структуры и ZeroMQ для обмена сообщениями. Реализация демонстрирует возможность создания и управления процессами, их взаимодействия через сообщения, а

также мониторинга их состояния. Использование ZeroMQ обеспечило высокую гибкость и масштабируемость системы. Работа иллюстрирует практическое применение системного программирования, асинхронного взаимодействия и структур данных для решения задач управления процессами.

Приложение

Src/tree.cpp

```
#include "../include/tree.hpp"

std::string Node::Ping(int _id) {
    std::string ans = "Ok:0";
    if (_id == id) {
        ans = "Ok:1";
        return ans;
    } else if (auto it = children.find(_id); it != children.end()) {
        std::string msg = "ping " + std::to_string(_id);
        SendMessage(it->second.get(), msg);
        if (auto msg_resp = ReceiveMessage(children[_id].get()); msg_resp.has_value()
&& *msg_resp == "Ok:1") {
            ans = *msg_resp;
        }
        return ans;
    }
    return ans;
}

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, ZMQ_REQ);

    int newPort = Bind(children[idChild].get(), idChild);
    childrenPort[idChild] = newPort;
    int pid = fork();

    if (pid == 0) { // ребенок
        execl(programPath.c_str(), programName.c_str(),
std::to_string(idChild).c_str(), std::to_string(newPort).c_str(), nullptr);
    } else { // родитель
        std::string pidChild = "Error: couldn't connect to child";
        children[idChild]->set(zmq::sockopt::sndtimeo, 3000);
        SendMessage(children[idChild].get(), "pid");
        if (auto msg = ReceiveMessage(children[idChild].get()); msg.has_value()) {
            pidChild = *msg;
        }
        return "Ok:" + pidChild;
    }
    return "";
}

std::string Node::Pid() {
    return std::to_string(getpid());
}

std::string Node::Send(const std::string& str, int id) {
    if (children.empty()) {
        return "Error: Not found";
```

```
    } else if (auto it = children.find(id); it != children.end()) {
        if (SendMessage(it->second.get(), str)) {
            std::string ans = "Error: Not found";
            if (auto msg = ReceiveMessage(children[id].get()); msg.has_value()) {
                ans = *msg;
                // Проверка, является ли команда 'kill'
                if (str.find("kill") == 0 && ans.find("Ok") == 0) {
                    // Удаление дочернего узла из списка
                    Unbind(children[it->first].get(), childrenPort[it->first]);
                    children[it->first]->close();
                    children.erase(it);
                    childrenPort.erase(id);
                    // std::cout << "Node " << id << " has been removed from
children." << std::endl;
                }
            }
            return ans;
        }
    } else {
        std::string ans = "Error: Not found";
        for (auto& child : children) {
            std::string msg = "send " + std::to_string(id) + " " + str;
            if (SendMessage(child.second.get(), msg)) {
                if (auto msg_resp = ReceiveMessage(child.second.get());
msg_resp.has_value()) {
                    ans = *msg_resp;
                    // Если получили положительный ответ, прекращаем цикл
                    if (ans.find("Ok") == 0) {
                        break;
                    }
                }
            }
        }
        return ans;
    }
    return "Error: Not found";
}

std::string Node::Kill() {
    // Для отладки
    // std::cout << "Node " << id << " sending kill command to all children." <<
std::endl;
    // std::cout << "Children before kill: ";
    // for (const auto& [child_id, socket] : children) {
    //     std::cout << child_id << " ";
    // }
    // std::cout << std::endl;

    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.has_value()) {
```

```
                msg = *tmp;
                // Ответ от дочернего узла
                if (!ans.empty()) {
                    ans += " " + msg;
                } else {
                    ans = msg;
                }
            }
        }
        // Kill обработан в Send
    }
    return ans;
}
```

Server.cpp

```
#include "../include/tree.hpp"
#include "../include/manage_zmq.hpp"
#include <fstream>
#include <signal.h>
#include <map>
#include <vector>
#include <sstream>
#include <algorithm>
#include <cctype>

std::string trim(const std::string& str) {
    size_t first = str.find_first_not_of(" \t\n\r");
    if (first == std::string::npos)
        return "";
    size_t last = str.find_last_not_of(" \t\n\r");
    return str.substr(first, (last - first + 1));
}

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) {
        std::string message;
        std::string command = "";
        if (auto msg = ReceiveMessage(&(task.parent)); msg.has_value()) {
            message = *msg;
        } else {
            // Если сообщение не получено, продолжить
            continue;
        }
        std::istringstream request(message);
```

```
        request >> command;

        if (command == "create") {
            int idChild;
            request >> idChild;
            std::string ans = task.Create(idChild, programPath);
            SendMessage(&task.parent, ans);
        } else if (command == "pid") {
            std::string ans = task.Pid();
            SendMessage(&task.parent, ans);
        } 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);
            if (!str.empty() && str[0] == ' ') {
                str.erase(0, 1); // Удаление ведущего пробела
            }
            std::string ans;
            ans = task.Send(str, id);
            SendMessage(&task.parent, ans);
        } else if (command == "exec") {
            int targetId;
            request >> targetId;

            // Чтение остальной части сообщения
            std::string execArgs;
            if (std::getline(request, execArgs)) {
                // Удаление ведущего '|', если есть
                if (!execArgs.empty() && execArgs[0] == '|') {
                    execArgs.erase(0, 1);
                }

                // Разделение на text и pattern по '|'
                size_t sep = execArgs.find('|');
                if (sep != std::string::npos) {
                    std::string text = execArgs.substr(0, sep);
                    std::string pattern = execArgs.substr(sep + 1);

                    // Удаление лишних пробелов
                    text = trim(text);
                    pattern = trim(pattern);

                    // Ограничение длины строк
                    if (text.length() > 108 || pattern.length() > 108) {
                        SendMessage(&task.parent, "Error: Strings exceed maximum
length of 108 characters");
                        continue;
```

```
                    }

                    // Логирование полученных данных (для отладки)
                    // std::cout << "Received exec command: ID=" << targetId
                    //           << ", Text='" << text << "', Pattern='" << pattern <<
"'" << std::endl;

                    // Выполнение поиска подстроки
                    std::vector<int> positions;
                    size_t pos = text.find(pattern, 0);
                    while(pos != std::string::npos) {
                        positions.push_back(static_cast<int>(pos));
                        pos = text.find(pattern, pos + 1);
                    }

                    // Формирование ответа
                    std::ostringstream response;
                    response << "Ok:" << targetId << ":";
                    if (!positions.empty()) {
                        for (size_t i = 0; i < positions.size(); ++i) {
                            response << positions[i];
                            if (i != positions.size() - 1) {
                                response << ";";
                            }
                        }
                    } else {
                        response << "-1";
                    }

                    // Логирование ответа (для отладки)
                    // std::cout << "Sending response: " << response.str() <<
std::endl;

                    SendMessage(&task.parent, response.str());
                } else {
                    SendMessage(&task.parent, "Error: Invalid exec command format");
                }
            } else {
                SendMessage(&task.parent, "Error: Invalid exec command format");
            }
        } else if (command == "kill") {
            std::string ans = task.Kill();
            ans = std::to_string(task.id) + " " + ans;
            SendMessage(&task.parent, ans);
            Disconnect(&task.parent, task.parentPort);
            task.parent.close();
            break;
        }
    }

    return 0;
}
```

Manage_zmq.cpp

```cpp
#include "../include/manage_zmq.hpp"


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(port);
        try{
            socket->bind(address);
            break;
        } catch(...) {
            port++;
        }
    }
    return port;
}
void Unbind(zmq::socket_t *socket, int port) {
    std::string address = "tcp://127.0.0.1:" + std::to_string(port);
    socket->unbind(address);
}

void Connect(zmq::socket_t *socket, int port) {
    std::string address = "tcp://127.0.0.1:" + std::to_string(port);
    socket->connect(address);
}

void Disconnect(zmq::socket_t *socket, int port) {
    std::string address = "tcp://127.0.0.1:" + std::to_string(port);
    socket->disconnect(address);
}

bool SendMessage(zmq::socket_t *socket, const std::string& msg) {
    try {
        zmq::message_t message(msg.size());
        memcpy(message.data(), msg.c_str(), msg.size());
        socket->send(message, zmq::send_flags::none);
        return true;
    } catch(const zmq::error_t& e) {
        std::cerr << "SendMessage error: " << e.what() << std::endl;
        return false;
    }
}

std::optional<std::string> ReceiveMessage(zmq::socket_t* socket) {
    zmq::message_t message;
    try {
        auto result = socket->recv(message, zmq::recv_flags::none);
        if (result && *result > 0) {
            return std::string(static_cast<char*>(message.data()), message.size());
        }
    } catch(const zmq::error_t& e) {
        std::cerr << "ReceiveMessage warning: " << e.what() << std::endl;
```

```
    }
    return std::nullopt;
}
```

Client.cpp

```
#include <set>
#include <iostream>
#include <sstream>
#include <memory>
#include "../include/tree.hpp"
#include "../include/manage_zmq.hpp"

// export PROGRAM_PATH="/workspaces/OS_MAI_Slobodin/build/lab5-7/server"

int main() {
    std::set<int> Nodes;
    std::string programPath = getenv("PROGRAM_PATH");
    Node task(-1);
    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()) {
                std::cout << "Error: Already exists" << std::endl;
            } else if (Nodes.find(idParent) == Nodes.end()) {
                std::cout << "Error: Parent not found" << std::endl;
            } else if (idParent == task.id) { // from -1
                std::string ans = task.Create(idChild, programPath);
                std::cout << ans << std::endl;
                Nodes.insert(idChild);
            } else { // from other node
                std::ostringstream strStream;
                strStream << "create " << idChild;
                std::string str = strStream.str();
                std::string ans = task.Send(str, idParent);
                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;
            } else if (task.children.find(idChild) != task.children.end()) {
                std::string ans = task.Ping(idChild);
                std::cout << ans << std::endl;
            } else {
                std::ostringstream strStream;
                strStream << "ping " << idChild;
                std::string str = strStream.str();
```

```
                std::string ans = task.Send(str, idChild);
                if (ans == "Error: Not found") {
                    ans = "Ok:0"; // Убрали пробел
                }
                std::cout << ans << std::endl;
            }
        } else if (command == "exec") {
            int id;
            std::cin >> id;
            if (Nodes.find(id) == Nodes.end()) {
                std::cout << "Error: Not found" << std::endl;
                continue;
            }

            std::string text, pattern;
            std::cin.ignore(); // Игнорируем оставшийся символ новой строки
            std::cout << "> "; // Запрос ввода text_string
            if (!std::getline(std::cin, text)) {
                std::cout << "Error: Failed to read text string" << std::endl;
                continue;
            }

            std::cout << "> "; // Запрос ввода pattern_string
            if (!std::getline(std::cin, pattern)) {
                std::cout << "Error: Failed to read pattern string" << std::endl;
                continue;
            }

            // Проверка длины строк
            if (text.length() > 108 || pattern.length() > 108) {
                std::cout << "Error: Strings exceed maximum length of 108 characters"
<< std::endl;
                continue;
            }

            // Формирование сообщения для отправки: "exec id|text|pattern"
            std::ostringstream msgStream;
            msgStream << "exec " << id << "|" << text << "|" << pattern;
            std::string execCommand = msgStream.str();

            // Отправка команды 'exec id|text|pattern'
            std::string ans = task.Send(execCommand, id);
            if (ans.empty()) {
                std::cout << "Error: No response from node" << std::endl;
                continue;
            }

            // Вывод ответа
            std::cout << ans << std::endl;
        } else if (command == "kill") {
            int id;
            std::cin >> id;
            std::string msg = "kill";
```

```
            if (Nodes.find(id) == Nodes.end()) {
                std::cout << "Error: Not found" << std::endl;
            } else {
                std::string ans = task.Send(msg, id);
                if (ans != "Error: Not found") {
                    std::istringstream ids(ans);
                    int tmp;
                    while (ids >> tmp) {
                        Nodes.erase(tmp);
                    }
                    ans = "Ok";
                    if (task.children.find(id) != task.children.end()) {
                        Unbind(task.children[id].get(), task.childrenPort[id]);
                        task.children[id]->close();
                        task.children.erase(id);
                        task.childrenPort.erase(id);
                    }
                }
                std::cout << ans << std::endl;
            }
        } else if (command == "exit") {
            std::cout << "Executing kill on client..." << std::endl;
            task.Kill();
            // std::cout << "Kill executed, exiting program." << std::endl;
            return 0;
        }
    }
}
```

Пример вывода:

```
root@c34508d80232:/workspaces/OS_MAI_Slobodin/build# ./lab5-7/client
create 10 -1
Ok:10881
create 20 10
Ok:10890
create 30 20
Ok:10902
create 40 30
Ok:10938
kill 20
Ok
ping 20
Error: Not found
ping 30
Error: Not found
```

```
ping 40
Error: Not found
ping 10
Ok:1
exec 10
> abracadabra
> bra
Ok:10:1;8
exit
Executing kill on client...
```